

Faculdade de Ciências da Universidade de Lisboa

Information Systems Project 2022-2023 Software Architecture Document (SAD)

**CONTENT OWNERS: Ana Teixeira fc56336, Daniel Lopes fc56357 e
João Vedore fc56311**

DOCUMENT NUMBER:

- 1

RELEASE/REVISION:

- 27

RELEASE/REVISION DATE:

- 26/11/2023

Table of Contents

1	Documentation Roadmap.....	8
1.1	Document Management and Configuration Control Information.....	8
1.2	Purpose and Scope of the SAD	8
1.3	How the SAD Is Organized	10
1.4	Stakeholder Representation	10
1.5	Viewpoint Definitions.....	11
1.5.1	Module View	12
1.5.1.1	Decomposition View.....	12
1.5.1.1.1	Decomposition Viewpoint Definition.....	12
1.5.1.1.1.1	Abstract.....	12
1.5.1.1.1.2	Stakeholders and Their Concerns Addressed	13
1.5.1.1.1.3	Elements, Relations, Properties, and Constraints.....	13
1.5.1.1.1.4	Language(s) to Model/Represent Conforming Views	13
1.5.1.1.1.5	Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria	13
1.5.1.1.1.6	Viewpoint Source	13
1.5.1.2	Data Model View	14
1.5.1.2.1.1	Data Model Viewpoint Definition	14
1.5.1.2.1.1.1	Abstract	14
1.5.1.2.1.1.2	Stakeholders and Their Concerns Addressed	14
1.5.1.2.1.1.3	Elements, Relations, Properties, and Constraints	14
1.5.1.2.1.1.4	Language(s) to Model/Represent Conforming Views.....	14
1.5.1.2.1.1.5	Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria	14
1.5.1.2.1.1.6	Viewpoint Source	14
1.5.1.3	Uses View	15
1.5.1.3.1	Uses Viewpoint Definition	15
1.5.1.3.1.1	Abstract.....	15
1.5.1.3.1.2	Stakeholders and Their Concerns Addressed	15
1.5.1.3.1.4	Language(s) to Model/Represent Conforming Views	15

1.5.1.3.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness	
Criteria	15
1.5.1.3.2 Viewpoint Source	15
1.5.2 C&C View	16
1.5.2.1 Client View.....	16
1.5.2.1.1 Client-Server Viewpoint Definition	16
1.5.2.1.1.1 Abstract.....	16
1.5.2.1.1.2 Stakeholders and Their Concerns Addressed	16
1.5.2.1.1.3 Elements, Relations, Properties, and Constraints.....	16
1.5.2.1.1.4 Language(s) to Model/Represent Conforming Views	16
1.5.2.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness	
Criteria	17
1.5.2.1.1.6 Viewpoint Source	17
1.5.3 Allocation View	17
1.5.3.1 Deployment View.....	17
1.5.3.1.1 Deployment Viewpoint Definition.....	17
1.5.3.1.1.1 Abstract.....	17
1.5.3.1.1.2 Stakeholders and Their Concerns Addressed	17
1.5.3.1.1.3 Elements, Relations, Properties, and Constraints.....	17
1.5.3.1.1.4 Language(s) to Model/Represent Conforming Views	18
1.5.3.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness	
Criteria	18
1.5.3.1.1.6 Viewpoint Source	18
1.6 How a View is Documented	18
2 Architecture Background	20
2.1 Problem Background	20
2.1.1 System Overview	20
2.1.2 Goals and Context	20
2.1.3 Significant Driving Requirements.....	20
2.1.3.1 Usability	20
2.1.3.2 Maintainability	21
2.1.3.3 Modifiability	21
2.2 Solution Background	22
2.2.1 Architectural Approaches	22
2.2.2 Analysis Results	22

2.2.3 Requirements Coverage.....	22
3 Views	24
3.1 Uses View	25
3.1.1 View Description.....	25
3.1.2 Primary Presentation.....	26
3.1.3 Element Catalog.....	26
3.1.3.1 Elements.....	26
3.1.3.2 Relations.....	27
3.1.3.3 Behavior	27
3.1.3.4 Constraints	28
3.1.4 Architecture background	28
3.2 Decomposition View.....	29
3.2.1 View Description.....	29
3.2.2 Primary Presentation.....	29
3.2.3 Element Catalog.....	30
3.2.3.1 Elements.....	30
3.2.3.2 Relations.....	31
3.2.3.3 Behavior	31
3.2.3.4 Constraints	31
3.2.4 Architecture background	31
3.3 Data Model View	32
3.3.1 View Description.....	32
3.3.2 Primary Presentation.....	32
3.3.3 Element Catalog.....	33
3.3.3.1 Elements.....	33
3.3.3.2 Relations.....	33
3.3.3.3 Behavior	33
3.3.3.4 Constraints	33
3.3.4 Architecture background	33
3.4 Client-Server View.....	34
3.4.1 View Description.....	34
3.4.2 Primary Presentation.....	34
3.4.3 Element Catalog.....	34

3.4.3.1	Elements	34
3.4.3.2	Relations.....	35
3.4.3.3	Behavior	35
3.4.3.4	Constraints	35
3.4.4	Architecture background	35
3.5	Deployment View.....	35
3.5.1	View Description.....	35
3.5.2	Primary Presentation.....	35
3.5.3	Element Catalog.....	36
3.5.3.1	Elements	36
3.5.3.2	Relations.....	36
3.5.3.3	Behavior	36
3.5.3.4	Constraints	36
3.5.4	Architecture background	36
4	Relations Among Views.....	37
4.1	General Relations Among Views.....	37
4.2	View-to-View Relations	38
5	SonarCloud	40
6	Referenced Materials	42
7	Directory	43
7.1	Index.....	43
7.2	Glossary	44
7.3	Acronym List.....	44

List of Figures

Figure 1: Usability Quality Scenario	21
Figure 2: Maintainability Quality Scenario	21
Figure 3: Modifiability Quality Scenario	22
Figure 4: Uses View	26
Figure 5: Decomposition View.....	29
Figure 6: Data Model View	32
Figure 7: Client-Server View	34
Figure 8: Deployment View.....	36

List of Tables

Table 1: Stakeholders and related criteria	11
Table 2: Stakeholders and Relevant Viewpoints	12
Table 3: Overview of Different Viewtypes, Elements, and Relationships in Software Architecture	25
Table 4: View-to-View Relations	38
Table 5: Materials.....	42
Table 6: Glossary	44
Table 7: Acronym List	44

1 Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 (“Document Management and Configuration Control Information”) explains revision history. This tells you if you’re looking at the correct version of the SAD.
- Section 1.2 (“Purpose and Scope of the SAD”) explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you’re seeking is likely to be in this document.
- Section 1.3 (“How the SAD Is Organized”) explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 (“Stakeholder Representation”) explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 (“Viewpoint Definitions”) explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 (“Views”). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.
- Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

- Revision Number: 27
- Revision Release Date: 26/11/2023
- Purpose of Revision: Final Revision
- Scope of Revision: All sections

1.2 Purpose and Scope of the SAD

This SAD specifies the software architecture for Information Systems Project 2022-2023 (PSI2223). All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system¹ is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. “Externally visible” properties refers to those assumptions other

¹ Here, a system may refer to a system of systems.

elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

This SAD is organized into the following sections:

- **Section 1 (“Documentation Roadmap”)** provides information about this document and its intended audience. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 (“Architecture Background”)** explains why the architecture is what it is. It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (Views”)** and **Section 4 (“Relations Among Views”)** specify the software architecture. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 (“Referenced Materials”)** and **6 (“Directory”)** provide reference information for the reader. Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

Table 1: Stakeholders and related criteria

	Reliability	Security	Performance	Useability	Availability	Understandability	Maintainability
Users	B	A	A	A	A	A	C
Acquirers	A	A	A	A	A	C	C
Security engineers	A	A	C	C	C	C	A
Application software developers	A	B	A	C	B	C	A
Application system engineers	A	B	A	C	A	<u>B</u>	A
Project Manager	A	B	A	A	A	A	A
Customer	A	A	A	A	A	A	C
Maintainer	A	A	B	B	B	B	A

Scale: A – big concern; B – some concern; C – a slight concern

1.5 Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then,

is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 2: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
Users	Client-Server
Acquirers	Deployment
Security engineers	Decomposition, Client-Server, Deployment
Application software developers	Decomposition, Uses, Client-Server, Data Model
Application system engineers	Decomposition, Uses, Deployment
Project Manager	Decomposition, Uses, Client-Server, Data Model
Customer	Client-Server
Maintainer	Decomposition, Uses, Client-Server

1.5.1 Module View

1.5.1.1 Decomposition View

1.5.1.1.1 Decomposition Viewpoint Definition

1.5.1.1.1.1 Abstract

The Decomposition viewpoint provides information on how the system is divided into units of implementation. This view describes how system responsibilities are partitioned across modules and how those modules are decomposed into sub-modules.

1.5.1.1.1.2 Stakeholders and Their Concerns Addressed

The stakeholders concerned with this viewpoint are the application software developers, application system engineers, the maintainers, project manager, security engineers, and performance analyst, since they are considered to need information on how the modules are divided, and what are the responsibilities of each module. The project manager must define work assignments. Understanding the system as a set of modules and submodules is very useful for that matter. Developers and maintainers need to have a good understanding of the responsibilities of each part of the system.

Security engineers, in particular, find value in this style as it helps in understanding how different modules interact and the potential security implications associated with each module. This understanding is vital for assessing and reinforcing security measures within the system. Application system engineers leverage this approach to comprehend the system's architecture and its breakdown into manageable components. This knowledge facilitates better decision-making when implementing and integrating new applications or systems into the existing environment.

1.5.1.1.1.3 Elements, Relations, Properties, and Constraints

The elements of this viewpoint are modules. The decomposition relation is a hierarchical relationship between modules, where a module can be decomposed into submodules. This relationship represents the "is-part-of" concept, indicating that submodules contribute to the overall functionality of the parent module. This particular viewpoint serves multiple purposes within software development. It aids in analyzing where modifications may occur within the system, facilitates the communication of the system's structure in manageable sections to new team members, and contributes valuable insights for task allocation.

Considerations like modifiability, choosing between building or reusing components, differentiating between common and unique parts in software product lines, and the skills of developers play vital roles in this decomposition process. This style of decomposition in software design helps in delineating responsibilities for modules, serving as a precursor to subsequent development phases. It also assists in conducting change and impact analyses and enables the effective distribution of work assignments among developers. Additionally, it aids in familiarizing newcomers with the software's organization by presenting it in comprehensible sections. Certain constraints govern the arrangement of these modules. Notably, the decomposition graph must not contain any loops, ensuring a clear and unambiguous hierarchy. Additionally, a crucial constraint dictates that each module can have only one parent, emphasizing a strictly hierarchical structure.

1.5.1.1.1.4 Language(s) to Model/Represent Conforming Views

Graphic and text concepts, described on the side, depending on the view, using UML as base.

1.5.1.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

The decomposition view assesses the modifiability of the system by understanding how modules are decomposed into submodules and conducts the impact analysis to determine the potential effects of changes in one module on others. For consistency and completeness, this view confirms that the hierarchy of modules and submodules is clearly represented, allowing stakeholders to easily understand the structure of the system.

1.5.1.1.1.6 Viewpoint Source

Theoretical Software Design Slides, Faculdade de Ciências da Universidade de Lisboa.

Documenting Software Architectures: Views and Beyond. Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, 2005.

1.5.1.2 Data Model View

1.5.1.2.1.1 Data Model Viewpoint Definition

1.5.1.2.1.1.1 Abstract

The data model viewpoint represents crucial information that requires storage within the system, playing an essential role in establishing a solid organizational structure for the data system.

1.5.1.2.1.1.2 Stakeholders and Their Concerns Addressed

The stakeholders involved in this viewpoint are:

- **Application software developers** play a crucial role in designing and implementing software modules that interact with the database. Their focus is on creating a modular view to ensure the well-organized data storage and management.
- **Project managers** make use of the data model view to guarantee that the data structure aligns with the project's goals and the data architecture remains adaptable to the project's requirements.

1.5.1.2.1.1.3 Elements, Relations, Properties, and Constraints

The elements of the data model view are data entities which in this project are users, games, ratings and opinions. These entities represent real world objects within a system and encapsulates specific information for the application. The elements can have one-to-one, one-to-many, many-to-many or many-to-one relations with other elements.

1.5.1.2.1.1.4 Language(s) to Model/Represent Conforming Views

The data model view is represented in UML notation.

1.5.1.2.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

The data model undergoes normalization analysis to eliminate data redundancy and ensure efficient data organization. Also, a rigorous schema validation process is applied to verify that the data model adheres to the project requirements. This view ensures integrity through well-defined relationships, preventing data inconsistencies. Each data entity includes attributes necessary for its purpose, leading to completeness.

1.5.1.2.1.1.6 Viewpoint Source

Theoretical Software Design Slides, Faculdade de Ciências da Universidade de Lisboa.

Documenting Software Architectures: Views and Beyond. Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, 2005.

1.5.1.3 Uses View

1.5.1.3.1 Uses Viewpoint Definition

1.5.1.3.1.1 Abstract

The uses viewpoint shows the developers what other modules must exist so the system works in an effective way. It shows the relations between modules, indicating how they should be used to maintain complexity under control, avoiding the system's modifiability degradation due to unwished dependencies.

1.5.1.3.1.2 Stakeholders and Their Concerns Addressed

The stakeholders involved in this viewpoint are:

- **Application software developers** are responsible for integrating modules within the application and ensuring that these modules interact effectively with each other.
- **Project managers** utilize the uses viewpoint to understand the module, helping in project planning. This viewpoint also helps reducing the project's complexity, leading to an easier management of the projects managers and guaranteeing the project is concluded within the time expected.
- **Maintainers.** The uses viewpoint helps maintainers identifying and fixing dependency problems and helps them making the necessary changes in the application without breaking functionality.
- **Application system engineers** use the Uses viewpoint to have a general view of the system.

1.5.1.3.1.3 Elements, Relations, Properties, and Constraints

The elements of the uses viewpoint are the modules of the application. The modules have uses/ depends-on relations with other modules. There are no constraints associated with this view, but it is important to note that loops are undesirable.

1.5.1.3.1.4 Language(s) to Model/Represent Conforming Views

The uses style is represented in UML notation.

1.5.1.3.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

- **Dependency Analysis:** Conduct the analysis of module dependencies to ensure that the uses/depends-on relations are well-defined and contribute to effective module interaction.
- **Modifiability:** Evaluate the impact of changes in one module on others, ensuring that the uses viewpoint supports modifiability.
- **Project Planning Enhancement:** Use this viewpoint to enhance project planning by providing insights into module relationships.
- **Dependency Clarity:** Verify that the dependencies between modules are clearly defined and documented, promoting a better understanding among developers and stakeholders.

1.5.1.3.2 Viewpoint Source

Theoretical Software Design Slides, Faculdade de Ciências da Universidade de Lisboa.

Documenting Software Architectures: Views and Beyond. Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, 2005.

1.5.2 C&C View

1.5.2.1 Client View

1.5.2.1.1 Client-Server Viewpoint Definition

1.5.2.1.1.1 Abstract

The client-server viewpoint offers a structured approach to manage shared resources and services in a distributed environment. It serves to separate client applications from service implementations, allowing for a clear and well-organized system design that promotes modifiability, scalability, security and other crucial factors.

1.5.2.1.1.2 Stakeholders and Their Concerns Addressed

This viewpoint involves several stakeholders, each with distinct concerns crucial to the successful operation of the system. These stakeholders include security engineers, developers, project managers and maintainers. **Security engineers** play a crucial role in ensuring the integrity, confidentiality and availability of the system, focusing on protecting it against several vulnerabilities and potential attacks, safeguarding the overall security of the system. **Application software developers** not only prioritize creating a structured and modular architecture so the system is effectively organized and supports scalability but also facilitating easier maintenance, updates and scalability. **Project managers** act like intermediaries between clients and servers, so they benefit from the clear separation of concerns. This separation enhances project planning, contributing to defining timelines for deliveries and budget constraints. **Maintainers**, responsible for code updates and bug fixes, appreciate well-organized system designs, being useful for modifications without causing unintended effects on the client side. **Customer** feedback often influences system improvements, making their experience more personalized and tailored to their preferences, thereby fostering loyalty and encouraging continued usage.

1.5.2.1.1.3 Elements, Relations, Properties, and Constraints

The elements of this view are the **clients**, representing a component that invokes the server's services, **servers**, component that provides services to the clients and the **request/reply connectors** that connects the client's invocations to the server and the server's responses to the client. The elements are connected through request/reply connectors, so the client is attached to the server with the request role and the server is attached to the client with the reply role. The constraints referred to this view are: the client and the server are connected to request/reply connectors and the server can be client to other servers.

1.5.2.1.1.4 Language(s) to Model/Represent Conforming Views

The client-server style is represented in UML notation.

1.5.2.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

- **Performance Analysis:** Assess the performance of the client-server architecture under different loads and scenarios to ensure optimal scalability.
- **Modifiability Analysis:** Evaluate the modifiability of the system by assessing how changes in client or server components impact the overall architecture, considering the concerns of application software developers.
- **Client-Server Relationship:** Confirm that the client and server elements maintain a clear and well-defined relationship through the request/reply connectors.
- **Security Guidelines Compliance:** Validate that the client-server architecture aligns with security guidelines and considers the requirements specified by security engineers.

1.5.2.1.1.6 Viewpoint Source

Theoretical Software Design Slides, Faculdade de Ciências da Universidade de Lisboa.

Documenting Software Architectures: Views and Beyond. Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, 2005.

1.5.3 Allocation View

1.5.3.1 Deployment View

1.5.3.1.1 Deployment Viewpoint Definition

1.5.3.1.1.1 Abstract

The decomposition view details the assignment of software components and their communication to the corresponding hardware elements.

1.5.3.1.1.2 Stakeholders and Their Concerns Addressed

The stakeholders that have concerns about this viewpoint are:

- **Security engineers:** This viewpoint is used for security, so the security engineers have concerns about ensuring the system is secure.
- **Application system engineers:** This viewpoint also concerns about the reliability of the system, so these stakeholders are properly concerned about the system's reliability.
- **Acquires:** are concerned about the cost of storing the objects in the hardware elements.
- **Project managers:**

1.5.3.1.1.3 Elements, Relations, Properties, and Constraints

The elements are the software elements: the processes from the C&C viewpoint and the environment elements: hardware elements. There are no constraints related to this viewpoint.

1.5.3.1.1.4 Language(s) to Model/Represent Conforming Views

The deployment style is represented in UML notation.

1.5.3.1.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

The Deployment View assess the dependencies between software components and hardware elements to ensure a clear understanding of the deployment relationships. It evaluates the cost implications of deploying software components on particular hardware elements to address concerns raised by acquirers regarding storage costs and verifies that the deployment configuration supports the reliability requirements outlined by application system engineers.

1.5.3.1.1.6 Viewpoint Source

Theoretical Software Design Slides, Faculdade de Ciências da Universidade de Lisboa.

Documenting Software Architectures: Views and Beyond. Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, 2005.

1.6 How a View is Documented

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter i stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter j stands for the number of the view packet being described: 1, 2, etc.

- Section 3.i.5.j: View packet #j.
- Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
 - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
 - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.
 - Section 3.i.5.j.2.3: Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
 - Section 3.i.5.j.2.4: Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
 - Section 3.i.5.j.2.5: Constraints. This section lists any constraints on elements or relations not otherwise described.
- Section 3.i.5.j.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5.j.4: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.5.j.5: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.
- Section 3.i.5.j.6: Relation to other view packets. This section provides references for related view packets, including the parent, children, and siblings of this view packet. Related view packets may be in the same view or in different views.

2 Architecture Background

2.1 Problem Background

The project that will be discussed corresponds to an evaluation for the course "Projetos de Sistema de Informações" in the Computer Engineering degree at the "Faculdade de Ciências da Faculdade de Lisboa".

This work was carried out over a semester within the same year. It is not expected to be compared with similar applications used globally because, even though it's completed, it doesn't present all the functionalities or security that the group intended to implement due to time constraints.

2.1.1 System Overview

This project aims to connect video game players in a specific location so they can share their favorite games by creating lists within their profiles. These profiles can be followed to access the shared games more quickly.

In terms of organization, the project is divided into "frontend" and "backend." The frontend focuses on the visual aspect and was developed using the Angular platform, which is based on TypeScript. It communicates via RESTful interactions with the backend to retrieve information from the database.

The backend is responsible for server-side logic, handling data manipulation within the database. It employs the Express framework, based on node.js, to create a RESTful API that manages the server's data logic. MongoDB is used as the server to store the data.

2.1.2 Goals and Context

The goal of this project is to serve as a meeting point for people who share an interest in video games. Therefore, a prototype has been created that allows users to create profiles, game lists, view games, among other functions. As it's nothing more than a prototype, our emphasis was on enhancing its functionalities rather than focusing extensively on its aesthetic quality.

2.1.3 Significant Driving Requirements

2.1.3.1 Usability

QAS1 - The user wants to create an account and start following a specific person already present in the database. This process can take between 2 to 5 minutes, depending on whether the name of the person on the platform is already known or not. Roughly, it takes about 1 minute to create an account and the remaining time to search for the specific person.



Figure 1: Usability Quality Scenario

2.1.3.2 Maintainability

QAS2 - If developers need to fix a bug in the front-end, they can do so without having to stop the site. Once done, the code is injected into the site, causing a brief refresh of the page to return to its normal operation.

However, if the bug is in the server logic, resolving it will require stopping the server and consequently the front-end page due to the closure of server information retrieval endpoints. Injecting the new code may take a varied amount of time, depending on the size of the database.

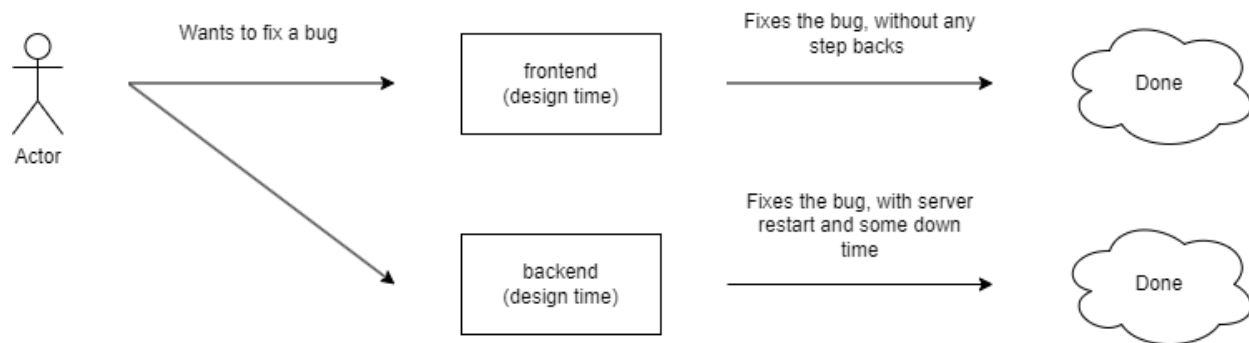


Figure 2: Maintainability Quality Scenario

2.1.3.3 Modifiability

QAS3 - If a programmer wants to add a new functionality, the behavior will be similar to the above scenario. In the front-end, there will be a brief refresh to implement the new code. However, in the back-end, it will require stopping the server for an indefinite amount of time, depending on the size of the database.

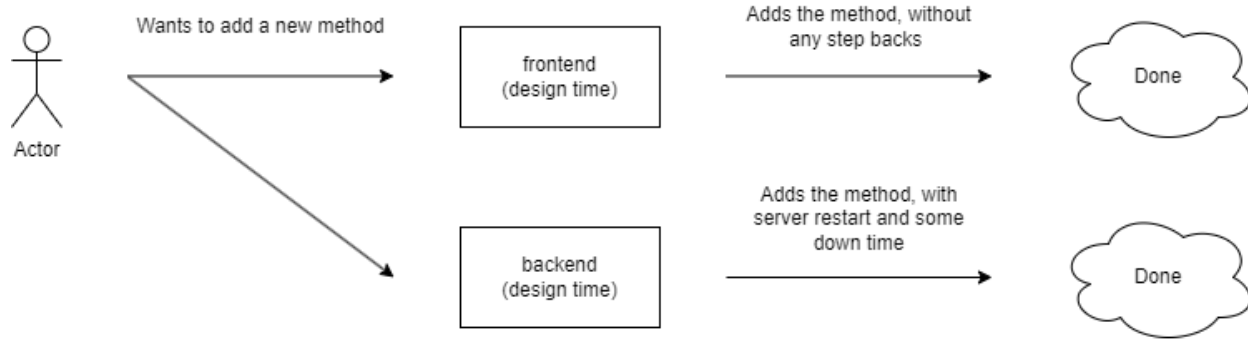


Figure 3: Modifiability Quality Scenario

2.2 Solution Background

2.2.1 Architectural Approaches

Due to the nature of the project's conception, there wasn't extensive architectural planning beyond the prior knowledge that the project should use the Angular framework for the frontend and Express with Node.js for the backend.

Nevertheless, it was calculated that in architectural terms, the work fits within the Client-Server aspect with 3 tiers.

The first tier comprises the user logic, responsible for displaying a view for the user, handling routing, and creating/storing cookies. This layer connects to the tier below through a RESTful relationship, obtaining and manipulating data.

The second tier corresponds to the server logic. As mentioned earlier, it handles data manipulation and extraction from the database. In other words, it manages primitive methods like "add," "update," "remove," as well as more complex data manipulation methods.

The third and final layer resides within the database, responsible for storing the data.

2.2.2 Analysis Results

As highlighted in previous topics, no tests were conducted in terms of performance using different architectures.

2.2.3 Requirements Coverage

During the development of this project, due to its nature, it was considered that the most important aspects would be modifying and reusing its code.

To illustrate this, let's consider the frontend. We can observe the presence of a class called "app.module.ts," which is responsible for all layers of the website. It's easy to create a new functionality and link it to this class. In terms of code reuse, this class serves as an example as it organizes all the application's functionalities.

Another requirement we focused heavily on was user usability. We aimed to create the best social experience among users, facilitating game sharing and visualization. This emphasis on user experience is clearly evident through the client interface.

3 Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- **Module views.** Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- **Component-and-connector views.** Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- **Allocation views.** These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Table 3: Overview of Different Viewtypes, Elements, and Relationships in Software Architecture

Name of view	Viewtype that defines this view	Types of elements and relations shown		Is this a module view?	Is this a component-and-connector view?	Is this an allocation view?
Uses	Uses viewpoint	Modules	Uses/Depends-on	Yes	No	No
Decomposition	Decomposition viewpoint	Modules	Is-part-of	Yes	No	No
Client-Server	Component-and connector viewpoint	Components, connectors	Attachment	No	Yes	No
Data model	Data model viewpoint	Entities	one-to-one, one-to-many, many-to-many or many-to-one	Yes	No	No
Deployment	Allocation viewpoint	C&C components	allocated-to	No	No	Yes

3.1 Uses View

3.1.1 View Description

This application is divided in frontend and backend packages. The frontend package has a component package which has all of the components/ modules in the frontend application, and the backend package has a models package with all the models/ modules of the backend application.

3.1.2 Primary Presentation

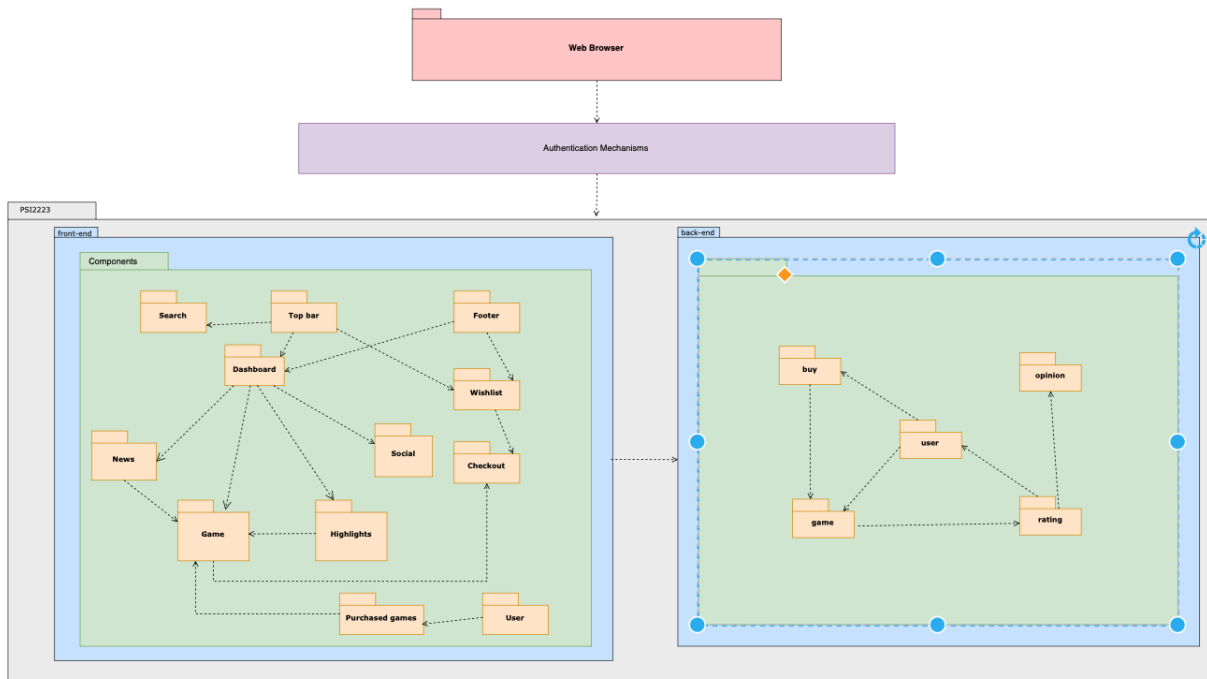





Figure 4: Uses View

Each arrow means a <<uses>> relation between the modules, so    means the module A uses the module B.

The colors of the packages don't mean anything, it's just to have a better visual perspective of the view.

3.1.3 Element Catalog

3.1.3.1 Elements

The elements of this view are the modules, which in this application are:

Frontend modules: correspond to the modules of the frontend application.

- **Top-bar:** is the top bar of the application and contains a searching bar and buttons to have a direct access to the user's wish list, the user's shopping cart and the user's profile.
- **Footer:** corresponds to the application's footer.
- **Dashboard:** it's the main page of the application. It has a direct view to the three latest released games, a view to the highlights of some games and the social bar. A button to access all the games is also on this element.
- **News:** this element contains the three latest released games.
- **Highlights:** this element contains some highlighted games.
- **Game:** represents the details of a game as well as its ratings, it has a direct way to put the game on the wish list and the shopping cart and has a direct access to the checkout.

- **Ratings:** a user can rate the game he is seeing on the application, rate them in a star scale and give their opinion as a comment.
- **Search:** the user can search a game in the top bar's search bar.
- **Wish list:** represents the games the user may want to buy.
- **Login:** is the initial page of the application. A user needs to put he's credential to access the system or may do a new login if he's new to the application.
- **Social:** used to check users' details.
- **User:** is the representation of a user. A user has he's profile with the games he already purchased, a list of the followers he has and a list of the users he's following.

Backend modules: correspond to the modules of the backend application.

- **Game:** represents the Schema of the game, having its details as well as its ratings, a direct way to put the game on the wish list and the shopping cart and a direct access to the checkout.
- **Ratings:** represents a rating of a user: a user can rate the game he is seeing on the application, rate them in a star scale and give their opinion as a comment.
- **User:** is the Schema representation of a user. A user has he's profile with the games he already purchased, a list of the followers he has and a list of the users he's following.
- **Opinion:** contains a textual opinion of a user.

3.1.3.2 Relations

The modules have several "uses" relations between each other.

3.1.3.3 Behavior

- The **top bar component** uses the wishlist component so the user can have a direct access to the user's wishlist, uses the shopping cart component as well to have a direct access to the user's shopping cart, uses the search component to have a direct access to the game the user wants to see and uses the user's profile component.
- The **dashboard component** uses the news component, the highlights component, the social component and the all games component.
- The **news component** uses the game to access to the three latest released games.
- The **highlights component** and the **all games component** use the game component to access all the games.
- The **game component** has a reference to the database to access its details, uses the ratings component, the wishlist component, the shopping cart component and the checkout component.
- The **shopping cart component** uses the checkout to buy the games that are in cart.
- The **purchased games component** uses the game component.

- The **user component** uses the purchased games component to access the games that were bought at the moment and uses itself to have a list with the user's follower and the users that he is following.
- **All components** use the top bar component and the footer component.

3.1.3.4 Constraints

The uses view has no constraints related to it. However, loops are unwished in the application.

3.1.4 Architecture background

This application is a game site that allows the logged users to see the available games in the site and buy them. The games are stored in a database, so the application is divided in two big modules: frontend and backend. The uses view and the diagram in 3.1.1.1.1. shows the general use of this application and the uses relations among the modules. The backend package has circularities because it encapsulates various interconnected functionalities necessary for managing and processing data related to games and user interactions.

3.2 Decomposition View

3.2.1 View Description

The frontend comprises user-facing modules like Dashboard, managing website navigation (Top_bar, all_Games, game_highlights, game_news, social, Login_page). It includes interactive elements such as Wishlist, Cart, and Search, while the Backend holds the logic and database interaction. Backend modules like Controllers handle user actions (buyController, gameController, ratingController, userController) and Models define the database structure (Buy, category, game, opinion, rating, type, user), ensuring data integrity and interaction.

3.2.2 Primary Presentation

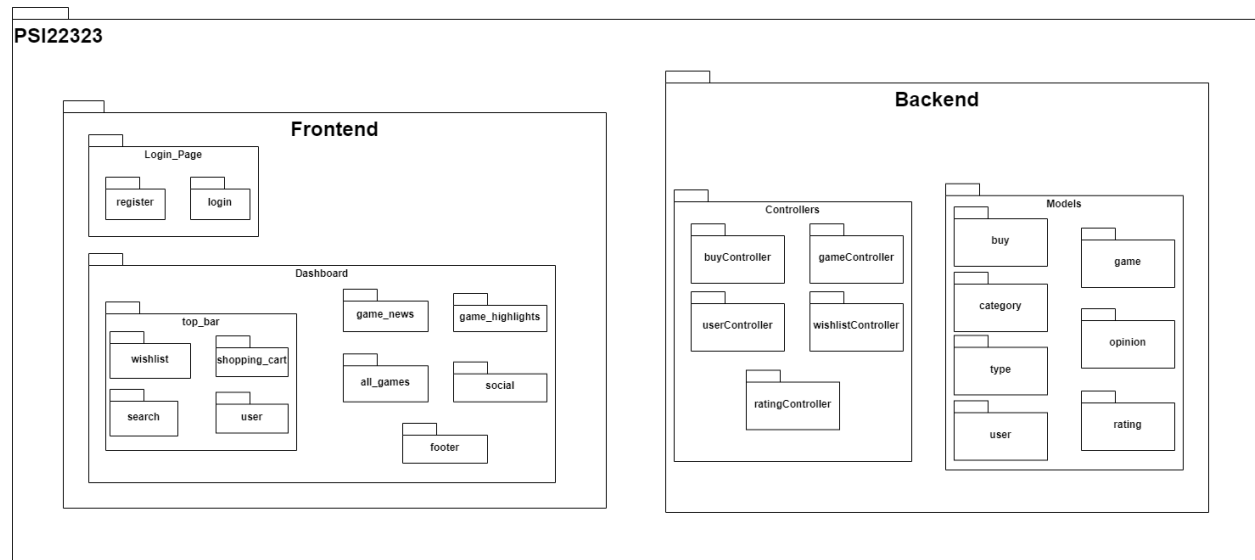


Figure 5: Decomposition View

3.2.3 Element Catalog

3.2.3.1 Elements

This section outlines the various modules within the website's architecture:

- Frontend: Module housing the website's front-end components.
 - Dashboard: Main page of the website.
 - Top_bar: Module present on all website pages, encompassing:
 - Wishlist: Enables control over games added to the wishlist.
 - Cart: Allows control over games added to the shopping cart and facilitates checkout if desired.
 - Search: Enables users to search for games of interest.
 - User:
 - all_Games: Module containing all games available on the website.
 - game_highlights: Module displaying the most purchased games.
 - game_news: Module showcasing the latest game releases.
 - social: Module facilitating the search for other users, following users, etc.
 - Footer: Module present across all website modules, displaying certain copyright-related information.
 - Login_page: Contains the authentication page for accessing the site.
 - Register: Allows users to create an account (user/password).
 - Login: Enables login with an existing account.
- Backend: Module housing the website's backend, managing requests, and interacting with the database.
 - Controllers: Module for controlling request for each data Module
 - buyController: Manages user purchases.
 - gameController: Controls all games on the website.
 - ratingController: Manages user ratings given to games.
 - userController: Manages all website users.
 - wishlistController: Controls all user wishlists.

- Models: Schemas of data models
 - Buy: Contains the buy schema.
 - category: Holds the category schema.
 - game: Includes the game schema.
 - opinion: Contains the opinion schema.
 - rating: Holds the rating schema.
 - type: Includes the type schema.
 - user: Holds the user schema.

3.2.3.2 Relations

The relations between the modules and submodules are is-part-of relations.

3.2.3.3 Behavior

The Frontend and the Backend are modules constitute the PSI2223 system. The Login_page and the Dashboard share a relation of is-part-of with the Frontend. The register and login is-part-of the Login_page. The Wishlist, Cart, Search and user share a relation of is-part-of with the Top_bar which has a relation of is-part-of with the Dashboard. The all_Games, game_highlights, game_news, social and Footer share a relation of is-part-of with the Dashboard.

The Controllers and the Models share a relation of is-part-of with the Backend. The buyController, gameController, ratingController, userController and wishlistController share a relation of is-part-of with the controllers.

The Buy, category, game, opinion, type, user and rating share a relation of is-part-of with the Models.

3.2.3.4 Constraints

The decomposition relation must adhere to the following constraints: Firstly, loops are strictly prohibited within the relation. Secondly, each module is limited to being a part of only one module and cannot be simultaneously involved in multiple modules

3.2.4 Architecture background

This application is a game site that allows the logged users to see the available games in the site and buy them. The games are stored in a database, so the application is divided in two big modules: frontend and backend.

3.3 Data Model View

3.3.1 View Description

This view shows a visual representation of the relations of the entities of the application. These entities are part of the backend package, contained within the models package.

3.3.2 Primary Presentation

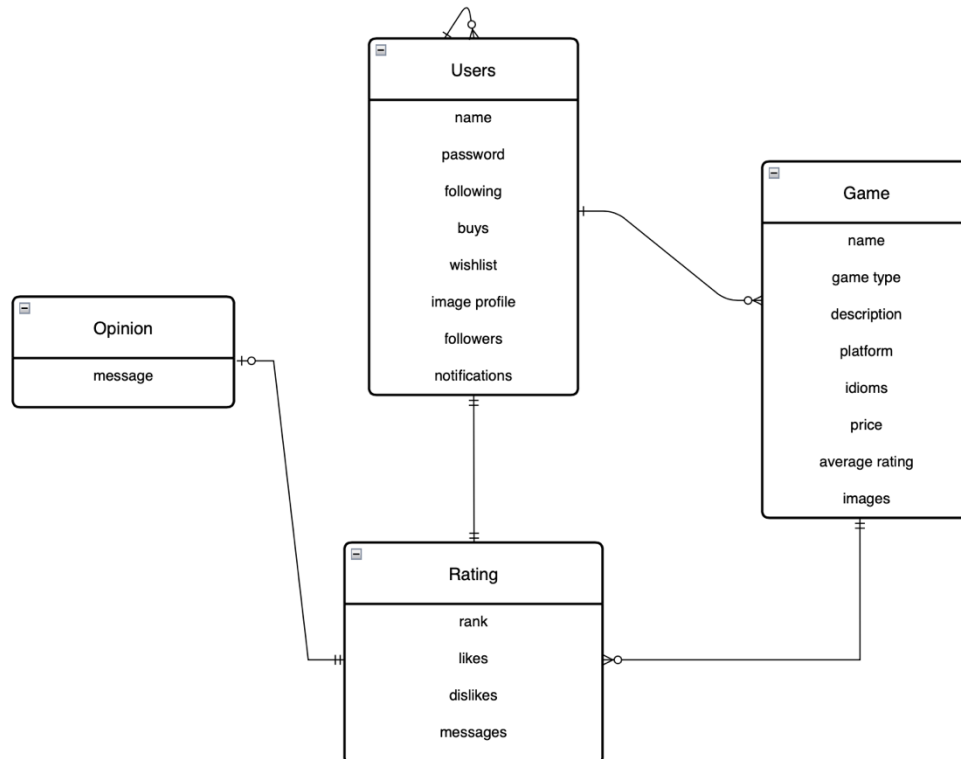
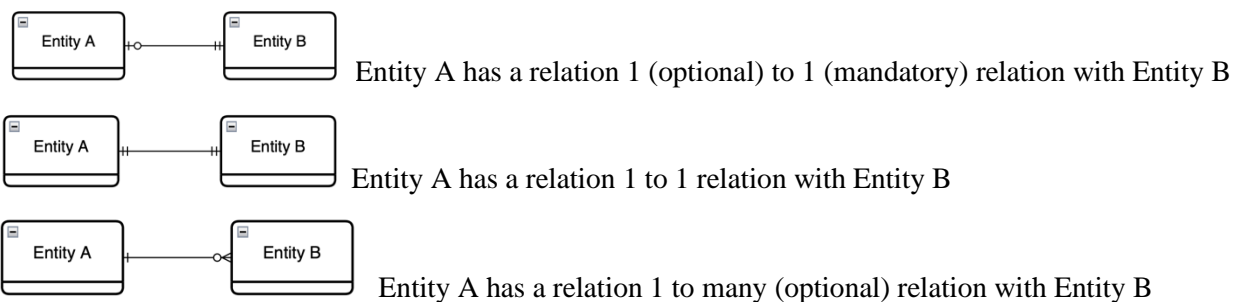


Figure 6: Data Model View

Labels:



3.3.3 Element Catalog

3.3.3.1 Elements

The elements of the data model view are data entities which in this application are almost all the models in the backend: users, games, ratings, opinions and buy. These entities represent real word objects within a system and encapsulates specific information for the application.

- **Game:** represents the Schema of the game, having its details as well as its ratings, a direct way to put the game on the wish list and the shopping cart and a direct access to the checkout.
- **Ratings:** represents a rating of a user: a user can rate the game he is seeing on the application, rate them in a star scale and give their opinion as a comment.
- **User:** is the Schema representation of a user. A user has he's profile with the gams he already purchased, a list of the followers he has and a list of the users he's following.
- **Opinion:** contains a textual opinion of a user.

3.3.3.2 Relations

The elements can have one-to-one, one-to-many, many-to-many or many-to-one relations with other elements.

3.3.3.3 Behavior

The entities have relations between each other:

- **Users:** have a relation one to one with games (a shopping cart and a wish list with references to many games) and other relation one to many with users ('followers' and 'following' both referring to other users).
- **Ratings:** relation one to one with the user and one to one with opinion.
- **Opinions:** relation one to one with user.
- **Games:** relation one to many with ratings.

3.3.3.4 Constraints

The data model view has no constraints related to it. However, loops are unwished in the application.

3.3.4 Architecture background

This application is a game site that allows the logged users to see the available games in the site and buy them. The games are stored in a database, so the application is divided in two big modules: frontend and backend. The data model view and the diagram in 3.3.1.1.1. shows the relation between the entities of the application, which are the models of the backend.

3.4 Client-Server View

3.4.1 View Description

The Client-Server view shows a representation of the interaction among the client and the server of the application.

3.4.2 Primary Presentation

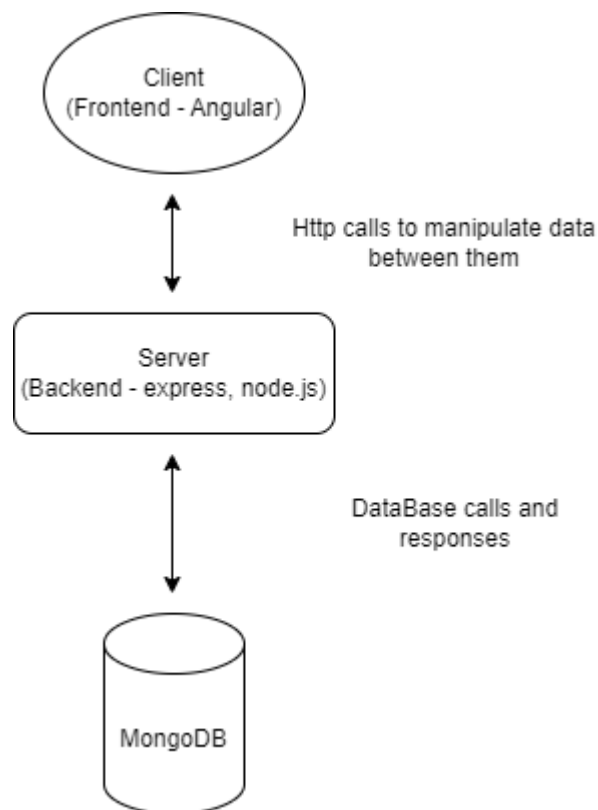


Figure 7: Client-Server View

3.4.3 Element Catalog

3.4.3.1 Elements

The elements of the client-server view are:

- Client: in this application the client corresponds to the Angular frontend.
- Server: corresponds to the backend of the application, in this case the express and node.js.
- Request/reply connectors.

3.4.3.2 Relations

The client server elements are connected through request/reply connectors.

3.4.3.3 Behavior

The frontend and the backend communicate through HTTP calls, so the frontend makes a request and the backend gets its response. The server is connected to the Mongo database to get the information of the application's games in order to serve/respond to the frontend

3.4.3.4 Constraints

The frontend and the backend are related through reply/request connectors.

3.4.4 Architecture background

This application has a client and a server that communicates through HTTP calls. The server communicates with the MongoDB, database where the game details are stored, to get the games information's needed.

3.5 Deployment View

3.5.1 View Description

This view shows a visual representation of how the components of the system are stored in the hardware elements.

3.5.2 Primary Presentation

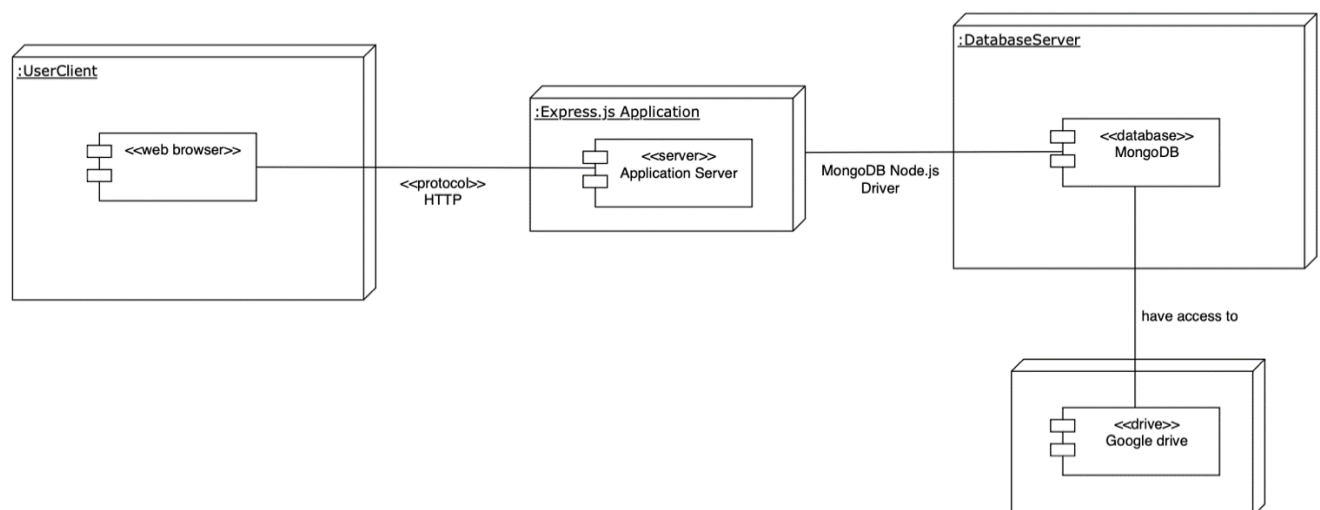


Figure 8: Deployment View

3.5.3 Element Catalog

3.5.3.1 Elements

The elements of the allocation view are:

- Software elements: the application server, as the game's information is stored in databases.
- Environment elements:
 - MongoDB: the database server that persists the games information.
 - Google drive: the drive that stores the images and the videos of the games.

3.5.3.2 Relations

The C&C component have an “allocated-to” relation with the environments.

3.5.3.3 Behavior

The server's component has an “allocated-to” relation with the MongoDB database and the Google Drive.

3.5.3.4 Constraints

There are no constraints related to this viewpoint.

3.5.4 Architecture background

This application refers to a site of games, so the games available need to be stored in a database which in this case is the Mongo database. Each game has at least one image and a video trailer, so in order to scale the application those images and videos are stored in a Google Drive.

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

The Decomposition view focuses on breaking down the system and highlighting the key modules, illustrating them without illustrating any form of dependency between them. This view aims to provide a clear structural understanding of the system's components.

On the other hand, the Uses view bears a resemblance to the Decomposition view but goes a step further by showcasing the dependencies between modules, delineating which modules are utilized by others. This view offers a more comprehensive insight into the interconnections and interactions within the system's architecture.

Both the Decomposition and Uses views are module-centric perspectives, where modules serve as the fundamental units of implementation, observed primarily from a design-time perspective.

Contrasting these module views, the Client-Server view introduces a different dimension by encapsulating the runtime behavior of entities (components) and their interactions (connectors). Despite referencing names already present in the module views, this perspective focuses on portraying the runtime behavior and interactions of components during the system's operation.

As the sole Allocation view, the Deployment view is tasked with illustrating how all the software units previously mentioned in the preceding views are deployed within the system's architecture. This view provides a comprehensive overview of how various software elements are allocated across hardware or computing resources to function cohesively as a system.

4.2 View-to-View Relations

Table 4: View-to-View Relations

Decomposition View	Deployment View
Module: Frontend	Software Elements
Module: Backend	Software Elements
Decomposition View	Uses View
Module: Frontend	Module: Front-end
Module: Backend	Module: Back-end
Decomposition View	Client-Server View
Module: Frontend	Component: Client
Module: Backend	Component: Server
Data Model View	Uses View
Entities: Users	Module: Back-end -> User
Entities: Game	Module: Back-end -> Games
Entities: Ratings	Module: Back-end -> Ratings
Entities: Opinion	Module: Back-end -> Opinion
Deployment View	Client-Server View
Software Elements	Component: Client, Server
Environment element -> MongoDB, Google drive	Component: Server
Entities: Ratings	Module: Backend-> Models -> rating
Entities: Opinion	Module: Backend-> Models -> opinion
Uses View	Client-Server View
Module: Front-end	Component: Client
Module: Back-end	Component: Server
Deployment View	Client-Server View
Software Elements	Component: Client, Server
Environment element -> MongoDB, Google drive	Component: Server

Data Model View	Deployment View
Entities: Users	Software Elements
Entities: Game	Software Elements
Entities: Ratings	Software Elements
Entities: Opinion	Software Elements

5 SonarCloud

The system underwent analysis using SonarCloud, an extension of GitHub, to assess its quality attributes. The tool evaluates Reliability, Security, Security Hotspots, and Maintainability on a grading scale from E to A.

5.1 Reliability:

- Rating: D
- Bugs (336):
 - Frontend (Angular - 73 bugs): These bugs might impact the user experience on the client interface, such as rendering issues, unresponsive interactions, or logic errors in the Angular frontend.
 - Backend (Node.js/Express.js - 263 bugs): Backend bugs could lead to server failures, like data handling errors, inadequate input validation, or security issues exploitable by malicious users.
- Severity: High (2), Medium (298), Low (36):
 - Issues with high severity typically hold greater priority and might pose serious threats to system stability, security, or functionality. Medium and low-severity bugs indicate less critical issues but still demand attention.

5.2 Maintainability:

- Rating: A
- Code Smells (202):
 - Frontend (Angular - 117 code smells): Code smells may pinpoint code areas that are hard to understand, maintained in a complex manner, or could be optimized for better readability and maintainability.
 - Backend (Node.js - 85 code smells): Maintenance issues in the backend might affect scalability, efficiency, or code readability, making it harder to maintain and expand.
- Severity: High (35), Medium (74), Low (93):
 - High severity code smells often signify more critical issues needing immediate attention, like hard-to-understand code, potential security flaws, or errors impacting future maintenance.

5.3 Security:

- Rating: E
- Severity: High (2):
 - Backend (MongoDB - 2): The recommendation to change and remove database passwords from the code is critical for security. These exposed passwords could lead to security breaches and unauthorized access to the database.

5.4 Security Review:

- Rating: E
- Security Hotspots (40):
 - Backend (37 Security Hotspots): This might indicate specific areas in the backend code requiring immediate attention due to potential security vulnerabilities.
 - Frontend (3 Security Hotspots): Additionally, there are three security hotspots identified in the frontend code that also demand immediate attention due to potential security vulnerabilities.

5.5 Duplications:

- Backend (3.2%); Frontend (1.7%):
 - Duplications: Despite relatively low percentages, code duplications can lead to unnecessary complexity, making maintenance difficult and introducing more opportunities for errors.

In conclusion, while the project shows strengths in Maintainability, there are critical areas requiring immediate attention in Reliability and Security. Resolving bugs, addressing code smells, handling security vulnerabilities, reducing duplications, and focusing on critical security points are vital steps toward enhancing the overall quality of project. These actions are essential for ensuring a more reliable, maintainable, and secure codebase, ultimately improving the user experience, and fortifying the system against potential threats.

6 Referenced Materials

Table 5: Materials

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. <i>Quality Attribute Workshops (QAWs)</i> , Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. < http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html >.
Bass 2003	Bass, Clements, Kazman, <i>Software Architecture in Practice</i> , second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, <i>Evaluating Software Architectures: Methods and Case Studies</i> , Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, <i>Documenting Software Architectures: Views and Beyond</i> , Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, <i>IEEE Recommended Practice for Architectural Description of Software-Intensive Systems</i> , 21 September 2000.

7 Directory

7.1 Index

Elements Names
Front-end (Uses View)
Back-end (Uses View)
Frontend (Decomposition View)
Backend (Decomposition View)
game (Data Model View)
ratings (Data Model View)
user (Data Model View)
opinion (Data Model View)
Server (Client-Server)
Client (Client-Server)
Request/reply (Client-Server)
Software Elements (Deployment View)
Environment Elements (Deployment View)

Relation Names
uses
is-part-of
One-to-many
One-to-one
many-to-many
Many-to-one
Request/reply connectors
Allocated-to

7.2 Glossary

Table 6: Glossary

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

7.3 Acronym List

Table 7: Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method

CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language