Uma imagem com Gráficos, Tipo de letra, captura de ecrã, logótipo

Descrição gerada automaticamente

Quizzes Tutor  
Software Architecture Document (SAD)

Leonardo Monteiro Nº58250

Henrique Vale Nº58168

Gustavo Henriques Nº64361

|  |  |  |
| --- | --- | --- |
| DOCUMENT NUMBER: | RELEASE/REVISION: | RELEASE/REVISION DATE: |
| * 1.0 | * Initial Release | * 28/10/2024 |
| * 1.1 | * First Update | * 5/11/2024 |
| * 1.2 | * Updates | * 07/11/2024 |
| * 1.3 | * More Updates | * 12/11/2024 |
|  |  |  |
|  |  |  |

Table of Contents

[1 Documentation Roadmap 3](#_Toc182322611)

[1.1 Document Management and Configuration Control Information 4](#_Toc182322612)

[1.2 Purpose and Scope of the SAD 4](#_Toc182322613)

[1.2.1 Purpose 4](#_Toc182322614)

[1.2.2 Scope 4](#_Toc182322615)

[1.3 How the SAD Is Organized 5](#_Toc182322616)

[1.4 Stakeholder Representation 6](#_Toc182322617)

[1.5 Viewpoint Definitions 7](#_Toc182322618)

[1.5.1 Module Viewpoint Definition 8](#_Toc182322619)

[1.5.1.1 Abstract 8](#_Toc182322620)

[1.5.1.2 Stakeholders and Their Concerns Addressed 8](#_Toc182322621)

[1.5.1.3 Elements, Relations, Properties, and Constraints 9](#_Toc182322622)

[1.5.1.4 Language(s) to Model/Represent Conforming Views 9](#_Toc182322623)

[1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 9](#_Toc182322624)

[1.5.1.6 Viewpoint Source 9](#_Toc182322625)

[1.5.2 Component-and-Connector Viewpoint Definition 10](#_Toc182322626)

[1.5.2.1 Abstract 10](#_Toc182322627)

[1.5.2.2 Stakeholders and Their Concerns Addressed 10](#_Toc182322628)

[1.5.2.3 Elements, Relations, Properties, and Constraints 10](#_Toc182322629)

[1.5.2.4 Language(s) to Model/Represent Conforming Views 10](#_Toc182322630)

[1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 10](#_Toc182322631)

[1.5.2.6 Viewpoint Source 10](#_Toc182322632)

[1.6 How a View is Documented 12](#_Toc182322633)

[1.7 Relationship to Other SADs 14](#_Toc182322634)

[1.8 Process for Updating this SAD 14](#_Toc182322635)

[2 Architecture Background 15](#_Toc182322636)

[2.1 Problem Background 15](#_Toc182322637)

[2.1.1 System Overview 15](#_Toc182322638)

[2.1.2 Goals and Context 15](#_Toc182322639)

[2.1.3 Significant Driving Requirements 15](#_Toc182322640)

[2.2 Solution Background 15](#_Toc182322641)

[2.2.1 Architectural Approaches 16](#_Toc182322642)

[2.2.2 Analysis Results 16](#_Toc182322643)

[2.2.3 Requirements Coverage 16](#_Toc182322644)

[2.2.4 Summary of Background Changes Reflected in Current Version 16](#_Toc182322645)

[2.3 Product Line Reuse Considerations 16](#_Toc182322646)

[3 Views 17](#_Toc182322647)

[3.1 <Insert view name> View 18](#_Toc182322648)

[3.1.1 View Description 18](#_Toc182322649)

[3.1.2 View Packet Overview 18](#_Toc182322650)

[3.1.3 Architecture Background 19](#_Toc182322651)

[3.1.4 Variability Mechanisms 19](#_Toc182322652)

[3.1.5 View Packets 19](#_Toc182322653)

[3.1.5.1 View packet # j 19](#_Toc182322654)

[3.1.5.1.1 Primary Presentation 19](#_Toc182322655)

[3.1.5.1.2 Element Catalog 19](#_Toc182322656)

[3.1.5.1.3 Context Diagram 19](#_Toc182322657)

[3.1.5.1.4 Variability Mechanisms 19](#_Toc182322658)

[3.1.5.1.5 Architecture Background 19](#_Toc182322659)

[3.1.5.1.6 Related View Packets 19](#_Toc182322660)

[4 Relations Among Views 20](#_Toc182322661)

[4.1 General Relations Among Views 20](#_Toc182322662)

[4.2 View-to-View Relations 20](#_Toc182322663)

[5 Referenced Materials 21](#_Toc182322664)

[6 Directory 22](#_Toc182322665)

[6.1 Index 22](#_Toc182322666)

[6.2 Glossary 22](#_Toc182322667)

[6.3 Acronym List 23](#_Toc182322668)

[7 Sample Figures & Tables 25](#_Toc182322669)

List of Figures

[Figure 1: Sample Figure 22](#_Toc51125206)

List of Tables

[Table 1: Stakeholders and Relevant Viewpoints 8](#_Toc87083002)

[Table 2: Sample Table 22](#_Toc87083003)

# 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
* **Section 1.7 ("Relationship to Other SADs")** notes whether this SAD is related to other architecture documents. If no related documents exist, it simply states "Not applicable," clarifying whether additional architecture documents should be consulted.
* **Section 1.8 (“Process for Updating this SAD”)** describes how to report issues or inaccuracies in the SAD, providing contact details and steps for handling feedback to maintain document accuracy.
* **TODO: Secalhar temos de pôr das restantes secções**

## Document Management and Configuration Control Information

* Revision Number: 1.3
* Revision Release Date: 12/11/2024
* Purpose of Revision: TODO
* Scope of Revision: TODO

## Purpose and Scope of the SAD

### Purpose

The purpose of this Software Architecture Document (SAD) is to comprehensively document the architecture of Quizzes Tutor, an open-source platform for creating, managing, and evaluating educational quizzes. This SAD serves as a central reference for the system’s development, analysis, and maintenance, supporting effective stakeholder communication and ensuring a shared understanding of its design.

### Scope

This SAD outlines key architectural aspects of Quizzes Tutor, including:

**Stakeholders and Their Interests**: Identification of main stakeholders (teachers, students, development team, and system administrators), highlighting their specific needs and concerns to guide architectural decisions.

**Quality Requirements**: Definition of key quality attributes such as performance, security, scalability, and maintainability, supported by scenarios illustrating expected system behavior and performance benchmarks.

**Architectural Views**:

* + **Module View**: Describes the modular structure and main responsibilities of each module, including the Vue.js frontend, Spring Boot backend, and PostgreSQL database.
  + **Component-and-Connector View**: Visualizes interactions and data flows between components, detailing how modules cooperate to handle user requests.
  + **Allocation View**: Maps software components onto the physical and virtual infrastructure, detailing deployment and resource management strategies.

This document focuses on high-level architectural decisions and component interactions, leaving detailed technical specifications and implementation guidelines to supplementary documentation. This approach provides a top-down view, making the system’s architecture accessible without delving into low-level code details.

## 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.

## Stakeholder Representation

This section provides an overview of the primary stakeholders involved in the development and use of the Quizzes Tutor system, along with their main concerns regarding the system’s architecture.

1. **Teachers**
   * **Role**: Creators and managers of quizzes.
   * **Main Concerns**:
     + Usability: An intuitive interface for creating and managing quizzes.
     + Reliability: System stability during usage to ensure smooth educational activities.
     + Performance Reporting: Access to reports tracking student progress and results.
2. **Students**
   * **Role**: End users who take quizzes.
   * **Main Concerns**:
     + Availability: Reliable access to the system at all times.
     + User Experience: A simple, user-friendly interface for completing quizzes.
     + Responsiveness: Fast response time, especially when submitting answers.
3. **Development Team**
   * **Role**: Engineers and developers responsible for system maintenance and expansion.
   * **Main Concerns**:
     + Modularity: An architecture that supports easy maintenance and future feature additions.
     + Scalability: The ability to handle an increase in users if necessary.
     + Documentation: Clear documentation to support ongoing development.
4. **System Administrators**
   * **Role**: Responsible for system deployment, stability, and monitoring.
   * **Main Concerns**:
     + Stability and Monitoring: Capability to monitor and resolve issues quickly.
     + Security: Protecting against unauthorized access and ensuring data integrity.
     + Resource Efficiency: Efficient management of server and storage resources.
5. **Project Manager**
   * **Role**: Supervisor ensuring project goals, timeline, and budget.
   * **Main Concerns**:
     + Schedule and Budget: Ensuring that the system is developed on time and within budget constraints.
     + Team Coordination: Facilitating collaboration among developers, designers, and stakeholders.
     + Architectural Flexibility: Ensuring that the architecture can adapt to changes or new requirements.

## 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.

For the Quizzes Tutor system, we have identified three primary viewpoints to capture essential system qualities and address the specific needs of stakeholders:

* **Module Viewpoint**: Focuses on the static structure, dividing the system into modules.
* **Component-and-Connector Viewpoint**: Highlights runtime interactions, showing data flow and communication.
* **Allocation Viewpoint**: Maps software elements to their physical or virtual deployment infrastructure.

The following table summarizes the stakeholders in the Quizzes Tutor project and the viewpoints selected to address their specific concerns:

Table 1: Stakeholders and Relevant Viewpoints

| Stakeholder | Viewpoint(s) that apply to that class of stakeholder’s concerns |
| --- | --- |
| Teachers | Module Viewpoint |
| Students | Component-and-Connector Viewpoint |
| Development Team | Module Viewpoint, Component-and-Connector Viewpoint |
| System Administrators | Component-and-Connector Viewpoint, Allocation Viewpoint |
| Project Manager | Module Viewpoint |

### Module Viewpoint Definition

#### Abstract

The Module Viewpoint organizes the Quizzes Tutor system into distinct modules, each representing a core function or service, such as user management, quiz handling, and data storage. This viewpoint provides stakeholders with a high-level view of the system’s structure and how responsibilities are distributed.

#### Stakeholders and Their Concerns Addressed

* **Teachers**: Need a clear view of core functionalities for educational use.
* **Development Team**: Benefits from a modular structure for simplified implementation and maintenance.
* **Project Manager**: Uses the module structure to organize work and track progress.

#### Elements, Relations, Properties, and Constraints

* **Elements**: Key modules include the frontend (Vue.js for UI), backend (Spring Boot in java), and database (PostgreSQL).
* **Relations**: Modules have “is-part-of” relationships in the system hierarchy, such as backend modules for quiz processing.
* **Properties and Constraints**: Each module should have distinct functions without overlap, to maintain independence and modularity.

#### Language(s) to Model/Represent Conforming Views

* **Modeling Language**: Represented with UML class or package diagrams showing module hierarchy and dependencies.

#### Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

* **Consistency**: Ensures that each functionality maps to a single module, avoiding redundancy.
* **Completeness**: Each module’s functionality covers its intended responsibilities, verified through requirements traceability.
* **Evaluation Techniques**: Scenario-based analysis (e.g., ATAM) to ensure that modules handle anticipated changes and growth.

#### Viewpoint Source

* Based on [Clements 2002, Section 2.1] for module decomposition principles.

### Component-and-Connector Viewpoint Definition

#### Abstract

The Component-and-Connector Viewpoint focuses on runtime interactions between components, such as data flow and communication protocols. This viewpoint is essential for understanding system behavior under various conditions and how components collaborate to deliver functionality.

#### Stakeholders and Their Concerns Addressed

* **Students**: Concerned with system reliability and response during quizzes.
* **Development Team**: Needs to understand component interactions for debugging and optimization.
* **System Administrators**: Monitors component interactions to maintain performance and resolve issues.

#### Elements, Relations, Properties, and Constraints

* **Elements**: Components include the frontend, backend APIs, and database connections.
* **Relations**: “Uses” and “connects-to” relations define interactions, e.g., frontend communicating with backend via RESTful APIs.
* **Properties and Constraints**: Each connection should ensure secure, high-availability communication.

#### Language(s) to Model/Represent Conforming Views

* **Modeling Language**: Represented using sequence diagrams or component diagrams to illustrate interactions.

#### Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

* **Consistency**: Ensures necessary connections exist without unnecessary duplication.
* **Completeness**: Each essential interaction for operations is mapped and validated.
* **Evaluation Techniques**: Load and performance testing to verify that components handle user demand effectively.

#### Viewpoint Source

* Based on best practices in runtime interaction modeling

### Allocation Viewpoint Definition

#### Abstract

The Allocation Viewpoint maps the Quizzes Tutor software components to physical or virtual infrastructure, detailing where each component operates. This viewpoint helps understand resource distribution and system scalability.

#### Stakeholders and Their Concerns Addressed

* **System Administrators**: Concerned with resource allocation and system scalability.

#### Elements, Relations, Properties, and Constraints

* **Elements**: Physical/virtual servers, cloud environments, and network connections.
* **Relations**: “Deployed-on” relationships showing where each software component is hosted.
* **Properties and Constraints**: Components should be distributed to optimize performance, with resources scaled to meet peak demand.

#### Language(s) to Model/Represent Conforming Views

* **Modeling Language**: Deployment diagrams to visualize component allocation to infrastructure.

#### Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

* **Consistency**: Ensures all components are deployed according to the architectural specifications.
* **Completeness**: Each software component is accounted for in the infrastructure map.
* **Evaluation Techniques**: Resource utilization analysis to optimize allocation and ensure scalability.

#### Viewpoint Source

* Based on deployment and infrastructure management best practices.

## How a View is Documented TODO (Esta parte foi 100% chat)

This section provides the structure and organization for documenting each view in Section 3 of the Software Architecture Document (SAD). If the format or layout in Section 3 is altered, it is essential to update this description accordingly to ensure consistency. This section acts as a standardized guide, which allows all views to follow a uniform documentation process, thus enhancing readability and comprehension across all stakeholder groups.

### Single-Presentation View Documentation

If a view’s documentation is presented in a single section without subdividing into view packets, the structure will be as follows:

* **Section 3.i: Name of view** – Specifies the view (e.g., *Module View*, *Component-and-Connector View*, etc.).
* **Section 3.i.1: View description** – Outlines the purpose and content of the view.
* **Section 3.i.2: Primary presentation** – Displays the main elements and their relationships using appropriate tools or notation.
* **Section 3.i.3: Element catalog** – Complements the primary presentation with further details on elements, relations, interfaces, behavior, and constraints.
* **Section 3.i.4: Context diagram** – Provides a diagram illustrating the context of the view within the system, marking its scope and interactions.
* **Section 3.i.5: Variability mechanisms** – Details any configuration or adaptation options within this view.
* **Section 3.i.6: Architecture background** – Explains rationale for major design decisions limited to this view.

### View Packet-Based Documentation

For views that require division into *view packets* to accommodate complexity, each view will be documented with the following structure:

* **Section 3.i: Name of view**
* **Section 3.i.1: View description** – Aligns with the viewpoint described in Section 1.5.
* **Section 3.i.2: View packet overview** – Lists all view packets within the view and explains why this set is comprehensive.
* **Section 3.i.3: Architecture background** – Describes relevant design decisions, requirements, and architectural choices specific to this view.
* **Section 3.i.4: Variability mechanisms** – Notes architectural mechanisms specific to this view, if applicable.
* **Section 3.i.5: View packets** – Each view packet follows the outline below:
  + **Section 3.i.5.j: View packet #j**
    - **Section 3.i.5.j.1: Primary presentation** – Depicts the core elements and their relationships.
    - **Section 3.i.5.j.2: Element catalog**:
      * **Elements** – Details the elements in the view packet.
      * **Relations** – Additional relations or constraints.
      * **Interfaces** – Visible software interfaces for each element.
      * **Behavior** – Any significant behavior of elements or interactions.
      * **Constraints** – Any specific constraints not covered earlier.
    - **Section 3.i.5.j.3: Context diagram** – Shows the scope and external interactions of this view packet.
    - **Section 3.i.5.j.4: Variability mechanisms** – Lists available configuration mechanisms.
    - **Section 3.i.5.j.5: Architecture background** – Rationale for design decisions specific to this packet.
    - **Section 3.i.5.j.6: Relation to other view packets** – References to related packets.

This organizational structure in Section 1.6 serves as a **template for Section 3**, ensuring consistency and clarity across all views in the SAD for the Quizzes Tutor project.

## Relationship to Other SADs

Not applicable.

## Process for Updating this SAD

Not applicable.

# Architecture Background

## Problem Background

The Quizzes Tutor system was developed to address specific needs in the educational field, particularly in simplifying quiz creation and management. This demand emerged from a gap in traditional assessment methods, which can be time-consuming and difficult to tailor for each class and subject. Quizzes Tutor aims to fulfill these needs by providing a practical, accessible, and secure digital platform.

### System Overview

The Quizzes Tutor project considers the following key constraints and influences on its architecture:

* **Flexibility and Ease of Use:** The system must allow teachers to create quizzes with various question types (e.g., multiple-choice, true/false) and provide an intuitive interface for both teachers and students.
* **Data Security:** Protecting user data and quiz results is a primary concern. The architecture must support data encryption and user authentication.
* **Performance and Scalability:** Given the potential for multiple simultaneous accesses, particularly during peak assessment periods, the architecture must support horizontal scalability and high availability.
* **Chosen Technologies:** The use of Vue.js for the frontend, Spring Boot for the backend, and PostgreSQL for the database influences architectural decisions, as each of these tools provides specific capabilities in modularity, performance, and security.

These combined factors form the background for the architectural choices made for Quizzes Tutor, ensuring the system effectively meets pedagogical and technical requirements.

### Goals and Context

This section outlines the main goals and contextual factors shaping the architecture of Quizzes Tutor. The system’s primary goal is to facilitate teaching and learning through interactive quizzes, ensuring that teachers can easily create and manage quizzes, while students have a seamless experience accessing and completing them. In the educational context, the system needs to be robust, focusing on usability, scalability, and security. The architecture of Quizzes Tutor plays a central role in the software lifecycle, aligning with system engineering artifacts and meeting both pedagogical and technical needs.

### Significant Driving Requirements

This section highlights the behavioral and quality requirements that drive the architecture of Quizzes Tutor. Performance requirements include the ability to handle high volumes of simultaneous requests, particularly during peak testing periods. Security requirements ensure that only authorized users have access to sensitive information. Other quality attributes, such as availability and maintainability, were evaluated using methods like ATAM(Architecture Tradeoff Analysis Method), ensuring the system can support future adaptations and enhancements without compromising data integrity or user experience.

## Solution Background

This section outlines the rationale behind the architecture chosen for Quizzes Tutor and explains how it meets the defined behavioral and quality objectives.

**Architecture Goal**: Designed to be a robust, scalable, and user-friendly platform for creating and managing educational quizzes, supporting both teachers and students. The system must be intuitive, reliable, and maintain consistent performance.

**Behavioral Goals**:

**User Experience**: The Vue.js frontend offers an interactive and responsive interface, crucial for a positive user experience in educational environments, where usability is key.

**Reliability**: Ensures consistent execution of core functionalities (e.g., quiz creation, submission, and grading), minimizing disruptions.

**Quality Attributes**:

**Performance**: The frontend-backend separation optimizes each component’s performance, keeping the interface responsive while the backend manages business logic and data processing efficiently.

**Scalability**: Using PostgreSQL and a modular implementation supports the system's growth for increasing users, while preserving integrity and performance.

**Maintainability**: The modular architecture facilitates feature additions and updates without impacting the entire application, enabling continuous improvement.

### Architectural Approaches

This section discusses the key design decisions underlying Quizzes Tutor’s architecture, including adopted architectural styles, patterns, and considered alternatives.

**Architectural Styles and Patterns**:

**Separate Frontend-Backend Architecture**: Vue.js is used for the frontend, separate from the Spring Boot (in java) backend, creating a responsive and scalable interface independent of business logic and data processing.

**Client-Server Model**: Sensitive operations and data are securely processed on the server (backend), while the client (frontend) offers an interactive user experience.

**Data Persistence with PostgreSQL**: Chosen for its reliability, scalability, and compatibility with complex transactions, PostgreSQL is ideal for managing quiz and assessment data.

**Design Rationale**:

**Vue.js for Frontend**: Chosen for its quick development cycle and easy learning curve, enabling the team to build an intuitive interface.

**Spring Boot for Backend**: Selected for robustness and modularity, managing business logic and integrating well with PostgreSQL.

**PostgreSQL as Database**: Ensures data integrity and consistency, essential for educational systems.

**Considered and Rejected Alternatives**:

**Frontend Frameworks**: React and Angular were considered, but Vue.js was chosen for simplicity and ease of integration.

**Databases**: MySQL and MongoDB were reviewed, but PostgreSQL was selected for handling complex transactions and ensuring data consistency.

**COTS Issues**:

**Use of Open-Source Solutions**: The architecture leverages open-source solutions, like PostgreSQL, to meet quality requirements without significant added cost.

**IST Authentication Integration**: Integrated with the Instituto Superior Técnico’s authentication system to ensure secure and straightforward access for users.

### Analysis Results

This section presents the results of quantitative and qualitative analyses conducted to assess the suitability of the "Quizzes Tutor" software architecture in meeting performance, security, and maintainability goals.

**Security Analysis**:  
The system has achieved a "Passed" status with minimal security concerns. There is 1 open security issue, resulting in an "E" rating. Additionally, 27 Security Hotspots have been identified, which require manual review to ensure they do not pose potential risks. While these hotspots do not directly indicate vulnerabilities, they mark areas that may benefit from further inspection to enhance the application's resilience against threats.

**Reliability Analysis**:

The system displays 9 open reliability issues, with a "D" rating in this metric. Although not severe, these issues suggest that some parts of the code could potentially lead to failures or unexpected behaviors. Resolving these issues would contribute to ensuring the application’s stability, particularly under high usage conditions.

**Maintainability Analysis**:  
There are 145 maintainability issues detected, yet the project maintains an "A" rating in this category. This indicates that, despite the identified issues, the code structure is generally sound and modular, supporting easier future modifications without adverse effects on other parts of the system. Addressing these maintainability issues could improve code clarity and modularity, ensuring long-term sustainability.

**Test Coverage**:  
The analysis revealed 0% test coverage out of a total of 10k lines of code. This lack of automated testing presents a considerable risk, as it leaves the system without a mechanism to consistently verify functionality and prevent regressions. Increasing test coverage should be prioritized to enhance the system's reliability and overall quality.

**Code Duplication**:  
The code has a duplication rate of 1.6% out of 41k lines analyzed. While this level of duplication is relatively low, reducing redundancy further would improve code readability and support long-term maintainability.

**Conclusion**:  
The analysis indicates that the "Quizzes Tutor" architecture is largely secure and maintainable, generally aligning with its intended goals. However, the system's reliability could benefit from improvements, and the absence of automated tests needs to be addressed to ensure consistent functionality and robustness.

### Requirements Coverage

This section summarizes how the Quizzes Tutor architecture meets both original and derived functional and quality requirements established for the project.

**Functional Requirements (Original)**:

**Quiz Creation and Management**: The modular architecture allows the backend to handle quiz creation and management independently, fulfilling the original requirement to support the educational process.

**User Interaction and Experience**: The Vue.js frontend ensures that the system is accessible and responsive, meeting the original requirement to provide an intuitive experience for students and teachers.

**Quality Requirements (Original and Derived)**:

**Security (Original)**: The architecture ensures data protection and controlled access through the integration of robust authentication mechanisms, meeting the original security requirement.

**Performance (Derived)**: The separation of frontend and backend, along with the choice of PostgreSQL, guarantees fast response times, addressing the derived requirement for performance even under heavy load.

**Scalability and Flexibility (Derived)**: The system’s modularity and the ability to add backend instances support user growth without significant reengineering, meeting the derived requirements for scalability and flexibility.

### Summary of Background Changes Reflected in Current Version

This section provides an overview of the architectural changes and their rationale since the initial release of the Software Architecture Document (SAD) for the Quizzes Tutor project. These changes have been influenced by ongoing analysis, requirement adjustments, and optimization opportunities identified through testing and stakeholder feedback.

**Improved Modularity**: Based on maintainability analysis with tools like SonarQube, certain parts of the code were refactored to improve modularity. This change enhances the ease of future updates and reduces interdependencies, allowing components to be updated independently.

**Enhanced Security Measures**: In response to stakeholder concerns and security assessments, the system now incorporates additional data protection measures, particularly in user authentication and data storage. This change was driven by the need to better protect sensitive information and align with updated security standards.

**Performance Optimization**: Performance testing revealed areas where system response times could be improved. As a result, database queries were optimized, and certain backend processes were restructured. These optimizations were necessary to ensure that the system can handle increased user load during peak times, enhancing scalability and user experience.

**Scalability Adjustments**: As part of the scalability strategy, the architecture has been updated to allow easier scaling by adding backend server instances as needed. This flexibility was essential to accommodate growing usage and was informed by load testing and anticipated growth.

These updates reflect a commitment to maintaining a robust and adaptive architecture that meets evolving project needs and aligns with quality standards in performance, security, and maintainability.

## Product Line Reuse Considerations

|  |
| --- |
| **CONTENTS OF THIS SECTION**: When a software product line is being developed, this section details how the software covered by this SAD is planned or expected to be reused in order to support the product line vision. In particular, this section includes a complete list of the variations that are planned to be produced and supported. "Variation" refers to a variant of the software produced through the use of pre-planned variation mechanisms made available in the software architecture. It may refer to a variant of one of the modules identified in this SAD, or a collection of modules, or the entire system or subsystem covered by this SAD. For each variation, the section identifies the increment(s) of the software build in which (a) the variation will be available; and (b) the variation will be used. Finally, this section describes any additional potential that exists to reuse one or more of the modules or their identified variations, even if this reuse is not currently planned for any increment. |

# Views

|  |
| --- |
| **CONTENTS OF THIS SECTION**: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5. |

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:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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?** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## <Insert view name> View

|  |
| --- |
| **CONTENTS OF THIS SECTION**: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view. |

### View Description

### View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

<<list, table, or diagram>>

### Architecture Background

### Variability Mechanisms

### View Packets

|  |
| --- |
| **CONTENTS OF THIS SECTION**: For each view packet in the view, this section describes it using the outline given in Section 1.6. |

#### View packet # j

##### Primary Presentation

##### Element Catalog

###### Elements

###### Relations

###### Interfaces

###### Behavior

###### Constraints

##### Context Diagram

##### Variability Mechanisms

##### Architecture Background

##### Related View Packets

# 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.

## General Relations Among Views

|  |
| --- |
| **CONTENTS OF THIS SECTION**: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified. |

## View-to-View Relations

|  |
| --- |
| **CONTENTS OF THIS SECTION**: For each set of views related to each other, this section shows how the elements in one view are related to elements in another. |

# Referenced Materials

|  |
| --- |
| **CONTENTS OF THIS SECTION**: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document. |

|  |  |
| --- | --- |
| Barbacci 2003 | Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, 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, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003. |
| Clements 2001 | Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001. |
| Clements 2002 | Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002. |
| IEEE 1471 | ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000. |

# Directory

## Index

|  |
| --- |
| **CONTENTS OF THIS SECTION**: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:  the location in the SAD where it was defined  each place it was used  Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location. |

## Glossary

|  |
| --- |
| **CONTENTS OF THIS SECTION**: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why. |

|  |  |
| --- | --- |
| 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. |

## 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 |

# Sample Figures & Tables



Figure 1: Sample Figure

Table 2: Sample Table

| Table Heading | Table Heading | Table Heading | Table Heading |
| --- | --- | --- | --- |
| Table Body | Table Body | Table Body | Table Body |
| Table Body | Table Body | Table Body | Table Body |
| Table Body | Table Body | Table Body | Table Body |
| Table Body | Table Body | Table Body | Table Body |

1. Appendices

|  |
| --- |
| **CONTENTS OF THIS SECTION**: Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data, API specification). As applicable, each appendix is referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. If your SAD has no appendices, delete this page. |

* 1. Heading 2 - Appendix
  2. Heading 2 - Appendix