Universidade do Minho
Escola de Engenharia
Master's Degree in Informatics Engineering

Course Unit: Software Requirements and Architectures

Academic Year 2025/2026

Proposal for an extension to the application PictuRAS

Rafael Fernandes (PG60298) Diogo Rodrigues (PG60244)
Paulo Ferreira (PG60288) Jorge Pereira (PG60276) Tiago Diogo (PG60308)

February 1, 2026

RAS

Abstract

This report documents the architectural solution implemented in the second development iteration of the *picturRAS* system, accompanying the delivery of the respective code. The document focuses on the architectural decisions made to support the introduction of new features, particularly the collaboration mechanisms associated with UC07, as well as corrections and improvements made to the existing system.

The characterization of the adopted architectural pattern and its influence on critical non-functional requirements, such as maintainability, scalability, and performance, is presented. Furthermore, decisions regarding the allocation of functional and non-functional requirements to solution components are described, grounded in relevant design principles and architectural tactics.

The report includes different architectural views, namely the *Building Block View*, the *Runtime Views* for collaboration and asynchronous cancellation, and the *Deployment View*, highlighting the system's organization and its mapping to the infrastructure. Finally, a critical analysis of the coverage of proposed requirements and tasks is presented, identifying potential deviations and limitations of the implemented solution.

Application Area: Software Architecture

Keywords: Software Architecture, Architectural Decisions, Non-Functional Requirements, Collaboration, Concurrency, Deployment

Index

List of Figures

List of Tables

1 Introduction

This technical report documents the architectural solution implemented within the scope of the second development iteration of the *picturRAS* system. This document accompanies the source code delivery and aims primarily to explain and justify the architectural decisions made to support the system's evolution, specifically the introduction of new collaboration features and the correction of limitations identified in the previous version.

The proposed solution results from the analysis of the existing architecture and its suitability for the new functional and non-functional requirements, with a particular focus on use case UC07, which introduces sharing mechanisms, collaborative sessions, and simultaneous editing. The need to ensure quality attributes such as maintainability, scalability, performance, and consistency led to the revision of the system's component organization and the adoption of specific strategies for concurrency management and asynchronous communication.

Throughout the report, the main architectural design decisions are presented and discussed, including the characterization of the adopted architectural pattern, the allocation of functional and non-functional requirements to solution components, and the application of relevant design principles and architectural tactics. Different architectural views are also described, namely the *Building Block View*, the *Runtime Views* associated with collaboration and asynchronous cancellation, and the *Deployment View*, allowing for an integrated understanding of the system's operation.

Finally, the report includes a critical analysis of the implementation carried out, assessing the degree of coverage of the proposed requirements and tasks and justifying potential deviations. Thus, it intends to provide a clear, grounded, and coherent view of the developed architectural solution.

2 Design Choices

2.1 Architectural Pattern

The architectural solution of the *picturRAS* system adopts a distributed service-oriented architecture, organized modularly and supported by communication gateways. Each service is responsible for a specific set of domain functionalities, such as project management, users, tools, subscriptions, and image storage.

The use of an *API Gateway* centralizes access to backend services, while a *WebSocket Gateway* is responsible for managing real-time communication, essential for supporting the collaboration mechanisms introduced in UC07. This separation allows for the isolation of functional concerns and facilitates system evolution.

The architecture is complemented by infrastructure components, such as an asynchronous message queue system (RabbitMQ) and an object storage service (MinIO), which contribute to the scalability, decoupling, and robustness of the solution.

2.2 Functional Allocation of Requirements to Components

Requirement	Description	Component(s)	New / Existing	Justification
RF-41	Real-time collaborative editing	*projects*, *wsGateway*, *apiGateway*	Existing	The editing logic remains in the projects service, while real-time synchronization is ensured by the *wsGateway*, specialized in bidirectional communication. This separation allows applying asynchronous cancellation mechanisms and performance optimizations without increasing coupling between components.
RF-45	Share projects via secure link	*users*, *subscriptions*, *projects*, *apiGateway*	Existing	Sharing management is allocated to the services responsible for identity and permissions, ensuring consistency with security and privacy mechanisms. This decision supports data encryption and immediate access revocation, respecting the separation of concerns principle.
RF-51	Permission validation	*subscriptions*, *projects*	Existing	Sharing links can be created with view or edit definitions. The project manager validates these permissions and ensures that the restrictions of each are applied.

Table 2.1: Functional allocation of requirements to solution components

Requirement	Description	Component(s)	New / Existing	Justification
RF-52	Link generation	*wsGateway*, *projects*, *apiGateway*	Existing	Collaboration link generation is integrated into the *wsGateway*, which already manages sessions and temporary access. This centralization ensures reduced response times and applies security policies associated with links, without duplicating logic in other services.
RF-53	Link access	*wsGateway*, *projects*, *frontend*	Existing	Link access is handled in the same component responsible for session management, ensuring high functional cohesion. This decision facilitates access error control and the application of clear and uniform messages in case of failure.
RF-54	Access to link management	*wsGateway*, *rabbitMQ*	Existing	Link management involves potentially asynchronous operations, such as invalidations and notifications, justifying the use of messaging. *rabbitMQ* reduces coupling between user requests and actual processing, supporting cancellations and event propagation.
RF-55	Revoke access	*imageStorageService*, *minio*	Existing	Access revocation implies ensuring that previously shared resources are no longer available. Responsibility is assigned to storage services, ensuring data consistency and support for the immediate revocation of links and sensitive data.

Table 2.2: Functional allocation of requirements to solution components (cont.)

Requirement	Description	Component(s)	New / Existing	Justification
RF-56	Link invalidation	*rabbitMQ*	Existing	Link invalidation is propagated asynchronously via messages, allowing immediate response to the user and consistent updates of all affected components. This approach supports security requirements and rapid cancellation of ongoing operations.
RF-57	Link error management	*apiGateway*	Existing	The *apiGateway* centralizes request validation and error handling, ensuring uniform and informative messages. This decision directly addresses the requirement for standardized error messages and increased system robustness.

Table 2.3: Functional allocation of requirements to solution components (cont.)

2.3 Allocation of Non-Functional Requirements

NFR	Quality Attribute	Description	Component(s)	Tactic / Justification
RNF-43	Performance	Application of common filters in under 5s and AI-based filters in under 1 minute, ensuring a fluid experience	*projects*, *wsGateway*, *rabbitMQ*	Asynchronous and decoupled processing via messages, avoiding interface blocking and supporting long-running operations. The *wsGateway* ensures real-time communication, while *rabbitMQ* allows for task scheduling and cancellation.
RNF-53	Concurrency / Consistency	Prevention of data corruption during simultaneous project edits	*projects*, *wsGateway*	Centralized project state management and real-time synchronization. The separation between business logic and communication reduces coupling and allows applying concurrency control strategies without impacting other services.
RNF-54	Scalability	Support for multiple users and concurrent processing	*rabbitMQ*, *wsGateway*	Use of messaging for load distribution and asynchronous execution. This tactic allows horizontally scaling processing and maintaining acceptable response times under load.

Table 2.4: Allocation of non-functional requirements to solution components

NFR	Quality Attribute	Description	Component(s)	Tactic / Justification
RNF-44	Security	Protection of access to shared projects via secure links	*apiGateway*, *subscriptions*, *wsGateway*	Centralized access and permission validation. Unique link generation and verification at the *apiGateway* ensure consistent access control and reduce sensitive data exposure.
RNF-52	Privacy	Guarantee of access only to authorized users, distinguishing between read and edit permissions	*users*, *subscriptions*, *projects*	Clear separation of concerns: the subscriptions service manages permissions, while projects only consumes access decisions. This approach ensures a high level of cohesion and privacy control.
RNF-19	Usability	Fluid interaction without the need for page reloading, with immediate user feedback	*wsGateway*, *apiGateway*	Real-time communication and immediate responses to the user. Centralizing request and error handling improves feedback clarity and user experience.

Table 2.5: Allocation of non-functional requirements to solution components (cont.)

3 Building Block View

3.1 Diagrama de Componentes

Despite the new additions to the project, these did not result in changes to the previously defined component diagram. They correspond only to variations at the behavioral and interaction levels between existing elements, neither introducing new microservices nor modifying responsibilities or dependencies between components. Thus, the microservices-based architecture with asynchronous communication remains unchanged, as it continues to satisfy the system's functional and non-functional requirements, particularly regarding scalability, elasticity, and parallel processing.

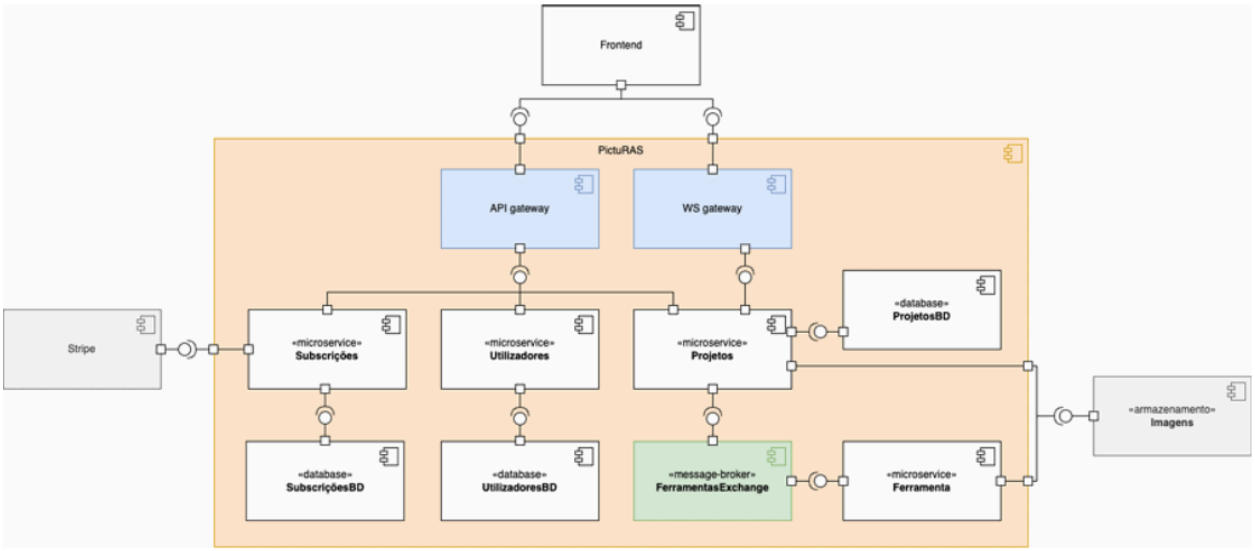

Figure 3.1: Component Diagram.

4 Runtime View – Collaboration (UC07)

4.1 Adding a Tool to a Project

The sequence diagram presented in Figure 4.1 illustrates the execution flow for adding or editing a tool within a project. This process involves orchestration between the Frontend, the Projects Service (API), the Users Service (Users MS), and the real-time messaging infrastructure.

The flow begins when the user (either the owner or a guest via a sharing *link*) submits a change in the interface.

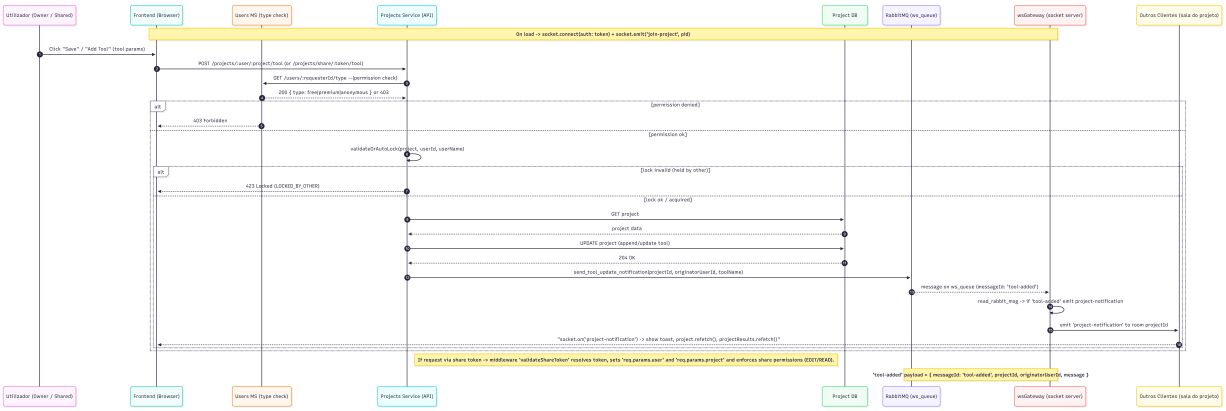

Figure 4.1: Tool Addition Sequence Diagram.

4.2 Concurrency Strategy

The system implements a robust concurrency control mechanism based on exclusive sessions (locks) with automatic renewal, complemented by real-time updates via WebSockets. This mechanism ensures data integrity when multiple users (Owners or Guests) access the same project simultaneously.

The core of the system lies in real-time updates via WebSockets to notify project state changes without the need for polling, and in the ***validateOrAutoLock*** function in the back-end (projects-ms), which is invoked prior to any write operation (e.g., adding tools, processing images, reordering) to ensure that no two changes are made at the same time.

Unlike traditional systems that require a manual click on "Edit", our approach is transparent ("Auto-Lock"):

- When a user attempts to modify an unlocked project, the system automatically assigns the lock to them.
- The project record in the database is updated with the ID of the user making changes, the acquisition timestamp, and the user's readable name.
- A WebSocket *lock-acquired* event is emitted to all connected clients, immediately updating the interface.

If a second user attempts to edit the project while it is locked, the operation is immediately rejected with an HTTP 423 Locked error, and the API Gateway propagates this error to the frontend, which notifies the user that the project is in use by another person.

5 Runtime View – Cancellation

5.1 Asynchronous Cancellation Propagation

To handle image processing cancellation, a button was implemented. When the user clicks it ("Cancel") during processing, the system activates a cancellation flag and immediately halts the pipeline. All queued processes are discarded, no new messages are sent to the tools, and the frontend is notified. As soon as a worker completes the currently executing filter, the processing stops.

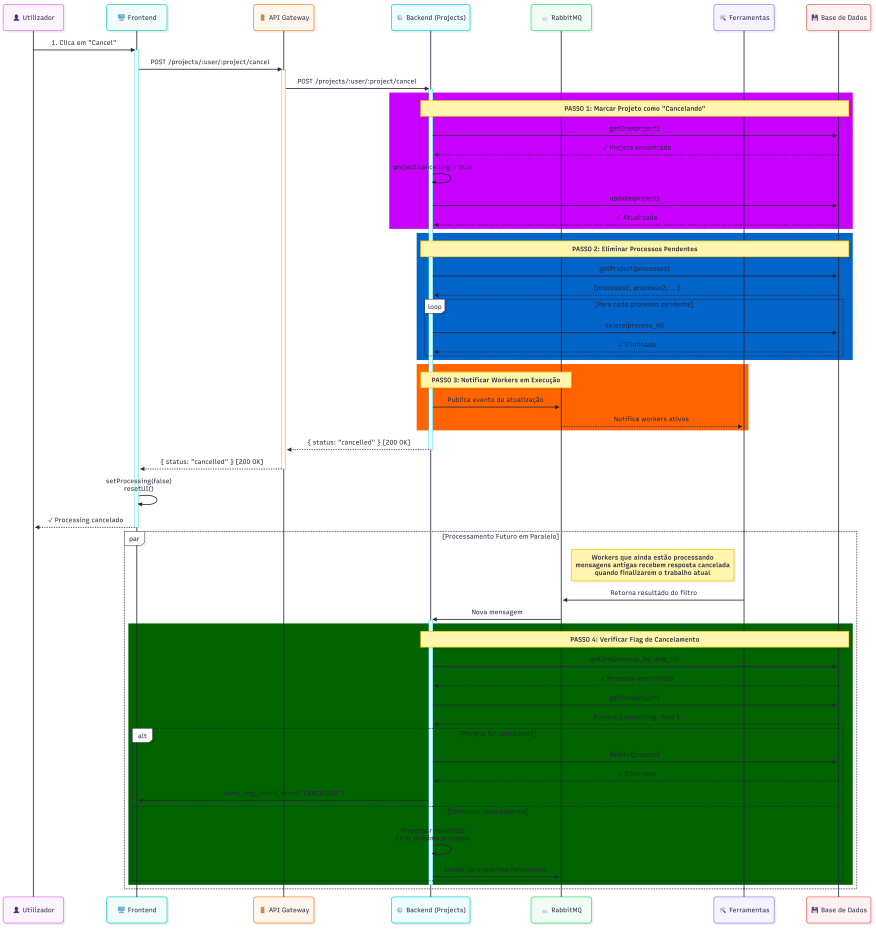

Figure 5.1: Cancellation Sequence Diagram.

6 Deployment View

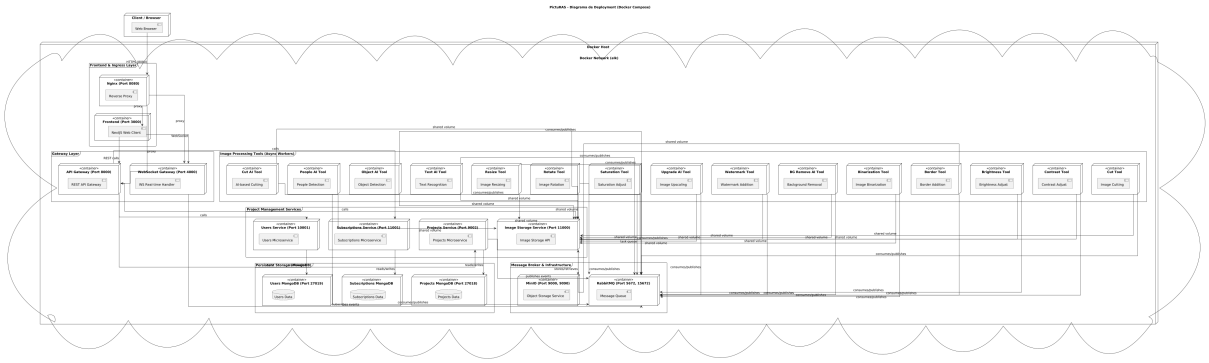

Figure 6.1: Deployment Diagram.

To create the containerized microservices system composing the PictuRAS application, approximately 28 containers are instantiated. Each service corresponds to a specific part of the infrastructure or application, and all interact through the `elk` network.

- **Nginx**: acts as a reverse proxy, receiving requests and forwarding them to internal services such as the `api_gateway`, `ws_gateway`, or `minio`.
- **MinIO**: used as object storage, similar to AWS S3, for storing images.
- **RabbitMQ**: serves as the message broker, allowing asynchronous services (such as image processing tools) to communicate via queues.
- **MongoDB**: multiple databases exist, one for each domain (`projects`, `users`, `subscriptions`), isolating data for each service.
- **Domain Services (microservices)**:
  - `projects`, `users`, `subscriptions` are microservices that manage projects, users, and subscriptions.
  - Each microservice depends on its own MongoDB and, in some cases, RabbitMQ.
- **Image Processing Tools** (`bg_remove_ai_tool`, `binarization_tool`, `resize_tool`, etc.): microservices that manipulate images. All depend on RabbitMQ to receive requests and use the `image_data` volume to read and write images.
- **Frontend**: connects to the backend and the API Gateway.
- **API Gateway and WS Gateway**: centralize HTTP and WebSocket requests, controlling authentication and communication with microservices.

All services use the `elk` network for internal communication, and volumes are utilized for data persistence (`project_data`, `user_data`, `image_data`, etc.), ensuring that information is not lost upon container restarts.

7 Critical Analysis

7.1 Coverage Table

Requirement	**Description**	**Implemented?**	**Observations / Justification of deviations**
RF41	Real-time collaborative editing	Partially	Structural project synchronization and main actions are implemented. However, continuous tool slider movement and real-time preview do not yet propagate instantaneous updates between users, due to current limitations in the real-time event mechanism.
RF45	Share projects via secure link	Yes	Unique links generated with permission levels, ensuring security and consistency with subscription management.
RF51	Permission Selection	Yes	User chooses between "View" or "Edit", with permission validation in the *apiGateway* and subscriptions.
RF52	Link Generation	Yes	Unique link generated immediately after permission selection, correctly encoding permissions.
RF53	Link Access	Yes	Recipient accesses the project according to permission (read-only or edit), with consistent behavior.

Table 7.1: Critical Analysis: Maintenance Tasks and UC07 Requirements Coverage

Requirement	Description	Implemented?	Observations / Justification of deviations
RF54	Access to Link Management	Yes	Active link management dashboard, allowing the owner to control shares and active links.
RF55	Revoke Access	Yes	Owner can revoke existing links, removing immediate access and ensuring data security.
RF56	Link Invalidation	Yes	Revoked links become unusable, with clear error messages upon access attempts.
RF57	Link Error Management	Yes	Uniform error messages displayed to the recipient accessing an invalid or expired link.
RNF-44	Security (access via secure link)	Yes	Centralized permission validation and unique link generation in the *apiGateway*, ensuring consistent access control.

Table 7.2: Critical Analysis: Maintenance Tasks and UC07 Requirements Coverage (cont.)

7.2 Justification of Deviations

Although collaborative editing is functional at the level of discrete actions, continuous updates of tool slider states do not yet occur in real-time between users, and the image preview on the guest user's side was not implemented. This deviation is due to limitations in event propagation through the currently implemented real-time communication mechanism and the lack of specific endpoints required to view the preview, with priority having been given to project state stability and consistency.

8 Conclusion

This report presented the architectural solution developed for the second iteration of the *pictur-RAS* system, detailing the main design decisions adopted to support the system's evolution and the introduction of new functionalities, with a particular focus on the collaboration mechanisms associated with use case UC07.

The adopted architectural pattern and the organization of the system's components were analyzed and justified, highlighting their influence on critical non-functional requirements, such as maintainability, scalability, performance, and consistency. The allocation of functional and non-functional requirements to the different components was grounded in design principles and, where applicable, in relevant architectural tactics, contributing to a more modular and extensible solution.

The different architectural views presented, namely the *Building Block View*, the *Runtime Views*, and the *Deployment View*, allowed for a clear description of the system structure, the execution flows associated with collaboration and asynchronous cancellation, as well as the mapping of components onto the infrastructure. These views contribute to a better understanding of the system's runtime behavior and its integration into the deployment environment.

Finally, the critical analysis performed allowed for the assessment of the coverage degree of the proposed requirements and tasks, identifying potential deviations and implementation limitations. Despite these limitations, the developed solution establishes a solid architectural foundation for the system's future evolution, allowing for the extension of functionalities and adaptation to new requirements with controlled impact on the existing architecture.