Software Architecture Overview

Fundamentals of Software Design: CS374

Antoine Gaton

October 13. 2024

Contents

[Background Information 3](#_Toc140505704)

[Functional Requirements 3](#_Toc140505705)

[Nonfunctional Requirements 3](#_Toc140505706)

[User Interfaces Specification 4](#_Toc140505707)

[Associations Between Requirements and Use Cases (Including UML Use-Case Diagrams) 4](#_Toc140505708)

[Associations Between Use Cases and User Interface 4](#_Toc140505709)

[Design Rationale 5](#_Toc140505710)

[Pros and Cons of Software Design Paradigm 1 5](#_Toc140505711)

[Pros and Cons of Software Design Paradigm 2 5](#_Toc140505712)

[Your Choice and Rationale 5](#_Toc140505713)

[References Page in APA Format 6](#_Toc140505714)

# Background Information

This document outlines the architecture of a new cloud-based, unified business contract management software system. The system is designed to replace the existing contract management systems built on Oracle DB, Microsoft SQL Server, and SAS Business Intelligence, serving a global company with over 50, 000 employees across 15 countries and 50 offices.

## Functional Requirements

1. Unified access to business contracts across all systems
2. Create and edit business contracts
3. Search and retrieve contracts

## Nonfunctional Requirements

1. Enhanced security for business data
2. Improved system availability and performance

# User Interfaces Specification

The system will provide a web-based interface accessible through standard web browsers, ensuring a unified experience across different devices and locations.

## Associations Between Requirements and Use Cases (Including UML Use-Case Diagrams)

The following UML case diagram illustrates the relationships between the system's requirements and use cases:

A diagram of a contract management system

Description automatically generated

The diagram shows two main actors interacting with the system:

1. *User*: Represents regular employees who interact with the contract management system.
2. *Admin*: Represents system administrators responsible for security and performance management.

The use cases are divided into two categories:

1. User-related use cases:
   * *Unified Access*: Allows users to access contracts from all integrated systems.
   * *Create/Edit Contracts*: Enables users to create new contracts and edit existing ones.
   * *Search/Retrieve*: Allows users to search for and retrieve specific contracts.
2. Admin-related use cases:
   * *Manage Security*: Enables administrators to implement and maintain security measures.
   * *Monitor Performance*: Allows administrators to monitor and optimize system performance.

This diagram illustrates how the functional and nonfunctional requirements are addressed through specific use cases, and how different user roles interact with these functionalities.

## Associations Between Use Cases and User Interface

* Contract Management: Web-based interface for contract creation, editing, searching, and retrieval
* System Administration: Separate administrative interface for security management and performance monitoring

# Design Rationale

## Pros and Cons of Software Design Paradigm 1

**Microservices Architecture**

Pros:

* Scalability: Easy to scale individual services independently
* Flexibility: Can use different technologies for different services
* Resilience: Failure in one service doesn't affect others
* Easier maintenance and updates

Cons:

* Complexity in managing distributed systems
* Potential performance overhead due to network communication
* Requires strong DevOps practices

## Pros and Cons of Software Design Paradigm 2

**Monolithic Architecture**

Pros:

* Simplicity in development and deployment
* Better performance for small to medium-sized applications
* Easier to test and debug

Cons:

* Less scalable for large applications
* Updates require redeploying the entire application
* Technology stack is uniform across the application

## Your Choice and Rationale

For this project, I have chosen the Microservices Architecture. The rationale for this decision is:

1. *Scalability*: With over 50,000 employees across 15 countries, the system needs to be highly scalable.
2. *Flexibility*: It allows for gradual migration from the existing systems (Oracle, Microsoft SQL, SAS) to the new unified system.
3. *Enhanced security*: Each microservice can have its own security measures, improving overall system security.
4. *Improved availability*: If one service fails, others can continue to function.
5. *Better performance*: Services can be optimized and scaled independently based on usage patterns.

# References Page in APA format

1. Eriksson, H. E. (2012). UML 2 toolkit. John Wiley & Sons.
2. Kaisler, S. H. (2005). Software paradigms. John Wiley & Sons.
3. Newman, S. (2015). Building microservices: designing fine-grained systems. O'Reilly Media, Inc.
4. Richards, M. (2015). Software architecture patterns. O'Reilly Media, Inc.