University of Sevilla

Higher Technical School of Computer Engineering

D01 – WIS Architecture Report

Degree in Computer Engineering - Software Engineering

Desing and Testing II

Course 2024 – 2025

Date	Version
18/02/2025	v1.0

Practice Group: C1.040	
Members	**Email**
Isabel Sánchez Castro	isasancas@alum.us.es
Javier Aponte Pozón	javapopoz@alum.us.es
José María Portela Huerta	josporhue@alum.us.es
Paula María Suárez Linares	pausualin@alum.us.es
Julia Virginia Ángeles Burgos	julangbur@alum.us.es

Repository: https://github.com/DP2-2025-C1-040/Acme-ANS

Table of contents

Revision Table

Date	Version	Description	Delivery
18/02/2025	V1.0	First Delivery analysis	D01

Introduction

In this document, we are going to share our knowledge about the architecture of a Web Information System (WIS), within the context of the course Design and Testing II.

We will discuss different types of architecture we know and their applications. It is important to note that this information is known to the group prior to the start of this course.

This provides a solid foundation for understanding and applying the concepts discussed in this course. Our goal is to provide an overview of the key concepts and best practices that we have learned in previous projects, thus enriching our learning process and encouraging discussion within the group and with the faculty.

Contents

Some of the architectures known to the group prior to the course are:

- **Layered architecture:**

  These architectures divide our application into various layers whose responsibilities are well distinguished. Each of these layers will offer a service to its lower and upper layers and will receive the service from another layer. The main benefit of this architecture is the low coupling and high cohesion due to the distinguished separation of responsibilities of each layer.

  However, one of the problems is that this architecture tends to be inefficient.
  One of the most common divisions is: a presentation layer, a business rules layer and a data access layer.

  Another advantage of this architecture is that most of the layers of the application can be hosted on our own servers, thus saving the client from having to spend resources to do all the internal work and simply interact with the data that our layers send to the Client or Presentation layer.

- **MVC architecture:**

  MVC is a pattern that focuses, like layered architecture, on the separation between business logic and its visualization.
  This separation provides a better division of work and improved maintainability.

  The three parts into which it is divided are:
  - **Model:** defines what data the application should contain.
  - **View:** defines how the application data should be displayed.
  - **Controller:** contains logic that updates the model and/or view in response to input from the application's users.

  The main difference with the layered model is the way the elements interact with each other.

  In the layered model, as we have discussed, each layer provides a service to its lower and upper layers, while in the MVC architecture it can be said that the Controller manipulates the Model, the Model is represented by the View and the View is used to call the Controller.

  Another benefit, apart from those already discussed, is that we can present multiple views using the same model and controller. While the main disadvantage is that usually requires a much higher learning curve than with other simpler architectures. simpler architectures.

- **Microservices Architecture:**

This framework has a more expansive approach unlike the other two architectures mentioned above. The main focus is the same, to have a well-defined separation of responsibilities. In this case what we do is to realize a small scale “application” that only offers an atomic service of our web application, thus obtaining multiple applications (which we will call microservices) whose responsibility is well defined and separated.

One of the greatest advantages of this architecture is that we can adapt the microservice to the requirements of each of these. We can use different technologies in each microservice, give more performance to some microservices, and so on.

On the other hand, the big disadvantage of this architecture is the handling, management and deployment of the microservices, and the learning curve can be even higher than the MVC architecture. In addition, this architecture only makes sense if our application is very large, since it would not be cost-effective to consume many resources for each microservice when the monolithic application can consume much less.

Conclusions

As a final conclusion, this document reflects our knowledge about several architectures that we can apply to the development of a web application. We have also seen advantages and disadvantages of these architectures and in which cases it is convenient to use each of them.

Bibliography

https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas

https://developer.mozilla.org/es/docs/Glossary/MVC

https://www.cloudmasters.es/sabias-que-que-son-los-microservicios-y-por-que-son-el-futuro-en-la-arquitectura-de-software/