D01 WIS Architecture Report

Deliverable 01

Group: C1.01.02

Repository: https://github.com/JMGarCas/Acme-L3

Student #1

ID: 77979757M
UVUS: enrcabmun
Name: Caballero Muñoz, Enrique
Roles: manager, developer, tester

Student #2

ID Number: 29574288Z
UVUS: felgudgue
Name: Gudiel Güemes, Félix Ángel
Roles: developer, tester

Student #3

ID Number: 45559252R
UVUS: josgarcas4
Name: García Casillas, José Miguel
Roles: developer, tester

Student #4

ID Number: 45996616C
UVUS: marchabar1
Name: Chasco Barry, Marco
Roles: developer, tester

Contents

The main source of information we have about WIS architecture comes from the previous course of Design and Testing, DP1. In this subject we learned about three architectural styles: layers, microservices and SPA (Single Page Application).

Layers

The layers architecture style is commonly used in applications with a user interface, where the presentation layer is separated from the business logic and data access layers. This separation allows each layer to be developed and maintained independently of the others, which can make the system more modular and easier to maintain over time.

There are typically three main layers in the layers architecture style:
-Presentation Layer: This is the layer that is responsible for interacting with the user. It includes components such as the user interface, input validation, and user authentication. The presentation layer communicates with the underlying layers, such as the business logic and data access layers, to retrieve and display data.
-Business Logic Layer: This layer is responsible for implementing the business logic of the system. It contains components that implement the rules and algorithms that govern how the system operates. The business logic layer interacts with the data access layer to retrieve and manipulate data.
-Data Access Layer: This layer is responsible for interacting with the data storage mechanisms, such as databases or file systems. It provides an interface for the business logic layer to retrieve and manipulate data.

Each layer in the layers architecture style should be designed with a specific set of responsibilities in mind. For example, the presentation layer should be designed to handle user input and provide feedback to the user, while the business logic layer should be designed to implement the rules and logic of the system. By separating these concerns into distinct layers, the system can be more flexible and easier to modify over time.

Microservices

Microservices is an architectural style that promotes the development of a software system as a collection of small, independent, and loosely coupled services. Each service is designed to perform a specific business capability and communicate with other services through well-defined APIs. This approach allows teams to work on individual services independently, enabling faster development and deployment cycles, and providing better scalability and flexibility.

In a microservices architecture, each service typically has its own database or storage system, which enhances the flexibility of the system. Modern communication protocols and technologies, such as RESTful APIs and message brokers, enable services to communicate in a decoupled and asynchronous manner, increasing the system's fault tolerance. The

microservices architecture also allows for scaling individual services independently, adapting to changing demands and ensuring resource optimization.
The microservices architecture style does present challenges, such as increased complexity and the need to ensure data consistency and integrity. For instance, the communication between services has to be thoroughly designed and tested to guarantee compatibility and correct functioning. In addition, monitoring and management of the multiple services and their interactions can be a complex task. By carefully weighing the pros and cons of the microservices architecture style and implementing the appropriate design and management strategies, organizations can create efficient, scalable, and reliable systems.

SPA

Single-Page Application (SPA) is a software architecture style for building web applications that dynamically update a single HTML page. This is done by loading the web application once, and then using JavaScript to update the content dynamically, without the need for a full page reload. In this architecture, the server-side code provides only the initial page and API endpoints to interact with the data, while the client-side code handles the application logic.

The main advantage of SPA is that it provides a faster, more responsive, and seamless user experience, as users can interact with the application without seeing the page refresh. Additionally, SPAs can reduce server load and bandwidth usage since only the necessary data is transferred between the server and the client. This is accomplished by leveraging modern web technologies such as AJAX, WebSockets, and JSON.

However, the SPA architecture style also presents some challenges, such as the difficulty of ensuring proper indexing and search engine optimization (SEO), since all content is on a single page. SPA applications are also heavily reliant on JavaScript, which can cause problems for users with outdated browsers or slow internet connections. Lastly, developers need to ensure that they optimize the application's performance and prevent memory leaks, as SPA applications can consume significant client-side processing power, leading to slow load times and decreased user experience.

Bibliography

Intentionally blank.