ACME AirNav Solutions



WIS ARCHITECTURE REPORT

Grupo: C1.050

Miembros: Cristina Fernández Chica ([criferchi@alum.us.es](mailto:criferchi@alum.us.es)), Ángel Amo Sánchez ([angamosan@alum.us.es](mailto:angamosan@alum.us.es)), Candela Jazmín Gutiérrez González ([cangutgon@alum.us.es](mailto:cangutgon@alum.us.es)), Marta Aguilar Morcillo ([maragumor@alum.us.es](mailto:maragumor@alum.us.es)) y Luis Emmanuel Chávez Malavé ([luichamal@alum.us.es](mailto:luichamal@alum.us.es))

Repositorio: <https://github.com/Cristinafernandezchica/Acme-ANS>

Planning dashboard: <https://github.com/users/Cristinafernandezchica/projects/1/views/1>

Sevilla 20 febrero, 2025

**ÍNDICE**

[**Introducción 4**](#_uqzfrhqx4u4)

[**Capa de Presentación (Frontend) 4**](#_eiaen27arzq7)

[**Capa de Lógica de Negocio (Backend) 4**](#_6azwzujdvdxj)

[**Capa de Persistencia (Base de Datos) 4**](#_fqvrkaw1wn3a)

[**Comunicación entre Capas 5**](#_y6njqurorvrt)

[**Registro y Monitoreo (Logging) 5**](#_v3t4td72mwpv)

[**Escalabilidad y Rendimiento 6**](#_1nrnhh9m018z)

[**Conclusión 6**](#_hq07xz5e0iit)

[**Bibliografía 6**](#_o5dkbw1z0jts)

#### **Introducción**

La arquitectura de un **Sistema de Información Web (WIS)** se basa en una estructura multicapa que separa las responsabilidades en diferentes niveles para mejorar la escalabilidad, el mantenimiento y la modularidad del sistema. Las capas principales son: la capa de presentación, la capa de lógica de negocio, la capa de persistencia y otros elementos que abordan el rendimiento y la seguridad del sistema.

#### **Capa de Presentación (Frontend)**

Esta capa se encarga de la interacción con el usuario y la visualización de la información de forma accesible. Se implementa utilizando tecnologías como HTML, CSS y JavaScript, y se complementa con frameworks como React, Angular o Vue.js. La capa de presentación se comunica con la capa de lógica de negocio a través de peticiones HTTP a APIs, permitiendo que el usuario interactúe con el sistema de manera dinámica. En aplicaciones más complejas, también se puede utilizar WebSockets para interactuar en tiempo real.

Buenas prácticas en el frontend incluyen separar las responsabilidades en los componentes, lo que facilita la organización del código, utilizar el Patrón Hook en React para gestionar la lógica de estado y los efectos secundarios de forma eficiente y practicar la componentización, creando componentes reutilizables para mejorar la mantenibilidad y evitar redundancias.

#### **Capa de Lógica de Negocio (Backend)**

El backend se encarga de procesar las solicitudes que provienen del usuario y gestionar toda la lógica del sistema. Esta capa se implementa con frameworks como Spring Boot (para Java), Django (para Python) o Express.js (para Node.js).

Dentro del backend, el sistema se organiza en tres elementos clave:

* Controller: Recibe las solicitudes del frontend, las valida y las pasa a la capa de servicio.
* Service: Contiene la lógica de negocio y las reglas que definen el comportamiento del sistema.
* Repository: Gestiona el acceso a la base de datos.

El uso de transacciones (@Transactional en Spring Boot) asegura que las operaciones en la base de datos sean consistentes y atómicas. Además, en caso de errores, se emplean excepciones personalizadas en el servicio, y el controlador utiliza @ExceptionHandler para manejar estos errores y devolver códigos HTTP adecuados.

#### **Capa de Persistencia (Base de Datos)**

Esta capa gestiona el almacenamiento y la recuperación de datos, utilizando bases de datos tanto relacionales (como MySQL y PostgreSQL) como NoSQL (como MongoDB). Se suele emplear un ORM (Object-Relational Mapping) como Hibernate o JPA para mapear objetos Java a tablas de la base de datos.

En la gestión de relaciones y asociaciones, se debe tener en cuenta la navegabilidad entre entidades, es decir, cómo unas entidades deben conocer a otras. Se utilizan estrategias de asociación como relaciones directas con atributos o clases de asociación en relaciones de muchos a muchos. Las consultas a la base de datos deben ser optimizadas para evitar dependencias innecesarias y evitar problemas como bucles infinitos en las respuestas JSON. Se aplican soluciones como @JsonIgnore en propiedades específicas y se implementan serializadores y deserializadores personalizados para evitar que se exponga información innecesaria a través de los JSON. Además, el uso de DTOs (Data Transfer Objects) ayuda a limitar la cantidad de datos enviados en las respuestas.

#### **Comunicación entre Capas**

La comunicación entre las distintas capas se realiza a través de peticiones HTTP, utilizando principalmente REST o GraphQL. En el caso de REST, el backend expone APIs con endpoints específicos como GET /pets para obtener una lista de mascotas o POST /owners para registrar un nuevo dueño. El frontend consume estas APIs mediante tecnologías como fetch() o Axios.

El Data Binding es esencial en esta capa para transformar datos entre el formato JSON y los objetos Java, un proceso que se realiza automáticamente en frameworks como Spring Boot. En aplicaciones que requieren comunicación en tiempo real, como chats o notificaciones en vivo, se pueden usar WebSockets.

**Seguridad y Gestión de Sesiones**

Para proteger el sistema, se implementan mecanismos de autenticación y autorización utilizando tecnologías como JWT (JSON Web Tokens) o OAuth2. Además, se protegen los endpoints mediante filtros de seguridad que se configuran en Spring Security. La gestión de sesiones en el frontend, como el almacenamiento de tokens en LocalStorage o SessionStorage, debe ser realizada con precaución para evitar problemas de seguridad. Para validar los datos en el backend, se usan anotaciones como @Valid, @NotNull y @Size. Además, las políticas de CORS (Cross-Origin Resource Sharing) pueden ser configuradas para restringir el acceso a las APIs solo a dominios específicos, añadiendo una capa extra de seguridad.

#### **Registro y Monitoreo (Logging)**

El monitoreo y registro de eventos en el sistema se realiza mediante herramientas como Logback en Spring Boot. Los logs pueden configurarse para tener diferentes niveles (INFO, DEBUG, ERROR, TRACE) y almacenar sólo la cantidad necesaria de registros. La configuración se puede personalizar a través del archivo application.properties. Además, se pueden implementar soluciones de monitoreo más avanzadas, como Prometheus o ELK Stack, para realizar un seguimiento del rendimiento y detectar posibles problemas de manera proactiva.

#### **Escalabilidad y Rendimiento**

La escalabilidad del sistema puede mejorar utilizando técnicas como el caching con Redis o Memcached para reducir la carga sobre la base de datos, así como la replicación de bases de datos para garantizar alta disponibilidad. En sistemas más grandes, se puede aplicar un balanceo de carga entre servidores para distribuir las peticiones y optimizar el rendimiento. En la capa de persistencia, el uso de sharding o particionamiento de bases de datos es útil para gestionar grandes volúmenes de datos.

#### **Conclusión**

En resumen, la arquitectura de un WIS sigue un enfoque multicapa que asigna una responsabilidad específica a cada capa del sistema. Este diseño modular facilita la escalabilidad, el mantenimiento y la separación de preocupaciones. La correcta gestión de la lógica de negocio, el manejo eficiente de los datos, la comunicación fluida entre el frontend y el backend, y las estrategias para asegurar el rendimiento y la seguridad son fundamentales para el éxito de un sistema web robusto y seguro.

#### **Bibliografía**

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4766122_1>

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4800078_1>

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4834440_1>

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4867971_1>

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4898721_1>

<https://ev.us.es/ultra/courses/_89406_1/outline/file/_4927556_1>