Logotipo

Descripción generada automáticamente

DOCUMENTO DAS ARQUITECTURA DE SOFTWARE PROYECTO

Documento de Arquitectura de Software IEEE-1471-2000 Control del documento



Integrantes:   
Francisco Baeza Hazael Díaz Mackarena Flores

• Nombre Proyecto: Cosmetología Morin

Nombre Sub Proyecto: Web Cosmetologa

Arquitectura Proyecto Semestral

Contenido

[Introducción 3](#_Toc214820498)

[Propósito 3](#_Toc214820499)

[Alcance 4](#_Toc214820500)

[Equipo de Trabajo-Actores del Desarrollo 5](#_Toc214820501)

[Recomendaciones de conformidad con esta práctica 6](#_Toc214820502)

[Referencias y estándares aplicables a este documento 7](#_Toc214820503)

[Definiciones, acrónimos y abreviaciones 7](#_Toc214820504)

[DESCRIPCIÓN DE ARQUITECTURA 9](#_Toc214820505)

[Framework Conceptual 10](#_Toc214820506)

[Actores Usuarios y sus roles 12](#_Toc214820507)

[Actividades de arquitectura en el ciclo de vida 12](#_Toc214820508)

[Descripciones prácticas de arquitectura 13](#_Toc214820509)

[Documentación de la arquitectura 15](#_Toc214820510)

[Planes de prueba 16](#_Toc214820511)

[Selección de puntos de vista de la arquitectura 17](#_Toc214820512)

[Vistas de la arquitectura 18](#_Toc214820513)

[Elementos Principales 21](#_Toc214820514)

[Relaciones 23](#_Toc214820515)

[Elementos Principales 24](#_Toc214820516)

[Relaciones 24](#_Toc214820517)

[Componentes del Proceso 27](#_Toc214820518)

[Servidor de Aplicaciones 29](#_Toc214820519)

[Servidor de Base de Datos 29](#_Toc214820520)

[Servidor de Notificaciones (Operación Manual) 29](#_Toc214820521)

[Navegación de Clientes y Trabajadores (Frontend Web) 29](#_Toc214820522)

[Centro de Datos Principal 31](#_Toc214820523)

[Red Local del Centro Cosmetología Morín 31](#_Toc214820524)

[Conexiones Externas 31](#_Toc214820525)

[Dispositivos de los Clientes 32](#_Toc214820526)

[Consistencia en la cantidad de vistas de la arquitectura 36](#_Toc214820527)

[Acceso a fuentes de información del proyecto 38](#_Toc214820528)

# Introducción

El presente informe tiene como objetivo describir en detalle la arquitectura del sistema web desarrollado para el proyecto Cosmetología Morín, una plataforma diseñada para automatizar la administración de tareas y procesos propios del negocio, tales como la gestión de reservas, productos, clientes y servicios.

El sistema propuesto busca mejorar la organización interna del emprendimiento, optimizar el flujo de atención de clientes y centralizar la información operativa, reduciendo la dependencia de canales manuales como WhatsApp y redes sociales.

El sistema desarrollado será implementado en un entorno web utilizando un servidor Node.js desplegado en Render, junto con una base de datos PostgreSQL alojada en la nube. Además, el sistema incorpora una interfaz intuitiva, responsive y accesible para todos los usuarios, asegurando que las gestiones se realicen de manera eficiente y estandarizada, siguiendo buenas prácticas de desarrollo, arquitectura modular y metodologías ágiles.

# Propósito

El propósito de este documento es servir como guía para el diseño e implementación de la arquitectura del sistema web desarrollado para Cosmetología Morín. En él se detallan las decisiones técnicas adoptadas, los componentes principales y los puntos de integración que permiten implementar una solución robusta, escalable y orientada a mejorar la administración del negocio.

Este informe es fundamental para los desarrolladores y stakeholders del proyecto, ya que proporciona una visión clara y detallada de la estructura y comportamiento del sistema, explicando cómo se relacionan sus funcionalidades y módulos internos. Asimismo, establece las bases para futuras expansiones y mejoras, considerando aspectos clave como la disponibilidad, la seguridad de la información, la mantenibilidad del código y la usabilidad de la plataforma.

# Alcance

El sistema gestionará las siguientes áreas:

* **Gestión de catálogo de productos y servicios**: Permite administrar la información de los servicios y productos ofrecidos, incluyendo imágenes, precios, descripciones y disponibilidad comercial.
* **Sistema de reservas y agenda de servicios**: Incluye un calendario interactivo para gestionar citas, visualizar disponibilidad y administrar estados de reserva (pendiente, confirmada o completada).
* **Gestión de clientes**: Registra información relevante de los clientes, su historial de servicios, pagos efectuados y datos de contacto, facilitando la trazabilidad de cada atención.
* **Panel administrativo**: Proporciona herramientas para gestionar inventario, servicios, productos, reservas, sesiones y reseñas desde un entorno centralizado diseñado exclusivamente para la administradora del negocio.
* **Módulo de reseñas y testimonios**: Permite recibir y gestionar opiniones de clientes, integrando un flujo de validación previa antes de su publicación en la plataforma.
* **Rol trabajador**: Ofrece acceso restringido para colaboradores, limitando su interacción a la visualización de la agenda y las citas del día, sin acceso a la configuración o administración del sistema.
* **Dashboards y analítica del negocio**: Incluye visualizaciones gráficas de métricas clave, tales como ventas mensuales y anuales, servicios más demandados e ingresos generados en diferentes períodos, permitiendo una toma de decisiones basada en datos.
* **Seguridad y administración del sistema**: Aplica control de roles, gestión de credenciales, medidas de protección básica de la información y un flujo de uso seguro dentro de la plataforma.
* **Infraestructura y despliegue**: El sistema se implementará en un entorno backend Node.js desplegado en Render, con una base de datos PostgreSQL en la nube, garantizando disponibilidad, escalabilidad y continuidad operativa.

# Equipo de Trabajo-Actores del Desarrollo

El desarrollo del sistema requiere la participación de un equipo multidisciplinario con roles claramente definidos para asegurar que el proyecto se lleve a cabo de manera eficiente. A continuación, se presenta una tabla con los principales roles y sus responsabilidades:

| Rol | Responsabilidades |
| --- | --- |
| Cliente-Stakeholder | Definir necesidades, validar avances, aprobar funcionalidades, entregar retroalimentación y asegurar que el sistema responda a los objetivos del negocio. |
| Product Owner | Priorizar el backlog, definir requerimientos, asegurar el alineamiento entre las necesidades del cliente y el desarrollo técnico; supervisar el cumplimiento de la visión del producto. |
| Desarrolladores Backend | Implementar la lógica del sistema utilizando Node.js, construir la API, conectar y gestionar la base de datos PostgreSQL, asegurar el correcto funcionamiento de los módulos internos y manejar integraciones. |
| Desarrolladores Frontend | Construir las interfaces de usuario, implementar vistas responsivas, conectar el frontend con la API backend y asegurar la usabilidad del sistema. |
| DBA-Especialista en Base de Datos | Diseñar el modelo de datos, crear y mantener la estructura en PostgreSQL, optimizar consultas, administrar relaciones y asegurar la integridad de la información. |
| Tester/QA | Ejecutar pruebas funcionales, validar módulos, detectar errores, verificar que los requisitos se cumplan y asegurar la calidad del sistema antes del despliegue. |
| Analista de Requisitos | Levantar información con la emprendedora, documentar requerimientos funcionales y no funcionales, definir alcance del sistema y apoyar la planificación del proyecto. |
| Administrador Plataformas | Encargado del despliegue en Render, configuración del entorno backend y base de datos, administración de versiones, control de hosting y monitoreo del funcionamiento del sistema. |

# Recomendaciones de conformidad con esta práctica

Para asegurar que la arquitectura del sistema cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas:

1. **Implementar revisiones de arquitectura de manera periódica** (idealmente cada sprint o cada 2 a 4 semanas), con el fin de asegurar que el diseño se mantenga alineado con los cambios de requisitos, nuevas funcionalidades y necesidades del negocio.
2. **Utilizar patrones de diseño desacoplados**, como arquitectura por capas o el modelo MVC, para mejorar la modularidad, escalabilidad y facilidad de mantenimiento del sistema. Esto permite reemplazar componentes sin afectar al resto de la plataforma.

1. **Adoptar metodologías ágiles como Scrum**, promoviendo la entrega continua de mejoras, revisiones constantes con la emprendedora y una adaptación flexible a necesidades emergentes durante el ciclo de desarrollo.
2. **Mantener una documentación técnica actualizada**, considerando diagramas, modelos de datos, rutas de API y flujos del sistema. Esto facilita el trabajo colaborativo, la corrección de errores y futuras expansiones del sistema.
3. **Realizar pruebas funcionales y de integración de forma constante**, asegurando la estabilidad del backend y la correcta interacción entre los módulos del frontend y la base de datos.
4. **Aplicar buenas prácticas de seguridad**, como la gestión adecuada de sesiones, control de roles, uso de SSL en Render y protección del acceso a la base de datos.
5. **Monitorear periódicamente el rendimiento del sistema y del hosting**, especialmente considerando que el proyecto puede escalar o replicarse para otros emprendedores del rubro.

# Referencias y estándares aplicables a este documento

El documento sigue los siguientes estándares de calidad para asegurar la robustez y escalabilidad del sistema:

* **IEEE 830-1998 (Software Requirements Specification – SRS)**

Utilizado como referencia para la estructuración y documentación de los requisitos del sistema, asegurando claridad, consistencia y verificabilidad de cada componente funcional y no funcional.

* **ISO 9126-2001 (Software Engineering – Product Quality)**

Considerado para evaluar atributos de calidad del software tales como funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad. Estos factores orientan las decisiones arquitectónicas tomadas durante el diseño del sistema.

* **Modelo 4+1 de Kruchten**

Empleado como marco conceptual para representar la arquitectura desde múltiples vistas (lógica, de procesos, de desarrollo, de despliegue y escenarios), facilitando la comprensión integral del sistema por parte de los distintos actores del proyecto.

* **ATAM (Architecture Tradeoff Analysis Method)**

Utilizado como guía para analizar decisiones arquitectónicas, identificando compromisos entre rendimiento, seguridad, mantenibilidad y escalabilidad, permitiendo documentar las razones detrás de cada elección técnica.

# Definiciones, acrónimos y abreviaciones

| Término | Definición |
| --- | --- |
| DAS | Documento de Arquitectura de Software |
| MVC / MODELO POR CAPAS | Modelo-Vista-Controlador, patrón de diseño utilizado para separar la lógica de negocio de la interfaz de usuario |
| UML | Unified Modeling Language, lenguaje gráfico para representar la arquitectura del sistema |
| CRUD | Operaciones básicas de Create, Read, Update, Delete para la gestión de datos. |
| API | *Application Programming Interface*; conjunto de endpoints y servicios que permiten la comunicación entre el frontend y el backend. |
| REST | *Estilo arquitectónico que define cómo deben estructurarse las APIs, basado en recursos, métodos HTTP y operaciones estandarizadas.* |
| Node.js | *Entorno de ejecución JavaScript utilizado para implementar el backend y la lógica del sistema.* |
| Express.js | *Framework para Node.js que facilita la creación de rutas, middleware y servicios backend.* |
| PostgreSQL | *Sistema de gestión de bases de datos relacional utilizado para almacenar la información del sistema.* |
| Render | *Plataforma cloud donde se despliega el backend del sistema y se mantiene la disponibilidad del servicio.* |
| Frontend | *Parte visual del sistema con la que interactúan los usuarios; implementada utilizando HTML, CSS y JavaScript.* |
| Backend | *Parte lógica del sistema encargado del procesamiento de datos, reglas de negocio y conexión con la base de datos.* |
| Dashboard | *Conjunto de gráficos e indicadores utilizados para visualizar datos relevantes del negocio, como ventas, servicios más solicitados e ingresos.* |
| Rol Trabajador | *Tipo de usuario con permisos limitados dentro del sistema, enfocado únicamente en visualizar la agenda y las citas asignadas.* |
| Stakeholder | *Persona o entidad interesada en el desarrollo y funcionamiento del sistema (en este caso, la dueña del negocio).* |
| Token / JWT | *JSON Web Token, mecanismo utilizado para gestionar autenticación y sesiones de manera segura.* |
| SSL | *Secure Sockets Layer, protocolo de seguridad utilizado para cifrar la comunicación entre el cliente y el servidor.* |

# DESCRIPCIÓN DE ARQUITECTURA

La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas.

**VISTAS:**

Cada vista arquitectónica del sistema será representada mediante diagramas UML, con el objetivo de describir de manera estructurada la organización, interacción y comportamiento de los componentes del sistema *Cosmetología Morín*. Estas vistas permiten comprender el sistema desde diferentes perspectivas, facilitando el análisis técnico, la comunicación con los stakeholders y la validación de las decisiones arquitectónicas.

En las siguientes secciones, se presentan las vistas definidas según el Modelo 4+1 de Kruchten, incluyendo diagramas que representan escenarios, estructura lógica, flujos de procesos, distribución de componentes y arquitectura física del sistema.

**TIPOS DE VISTAS:**

La arquitectura del sistema se estructura utilizando las vistas definidas en el Modelo 4+1 de Kruchten, permitiendo analizar la solución desde diferentes perspectivas técnicas y funcionales. Cada vista se representa mediante diagramas UML que facilitan la comprensión de los componentes y su interacción dentro del sistema *Cosmetología Morín*. A continuación, se describen las vistas consideradas:

* **Vista de Escenarios (Diagrama de Caso de Uso):** Describe las funcionalidades del sistema desde la perspectiva del usuario, representando cómo interactúan los diferentes actores —administradora, trabajador y cliente— con los principales casos de uso del sistema, tales como gestionar reservas, visualizar servicios, administrar productos o acceder al panel administrativo.
* **Vista Lógica (Diagrama de Clases):** Detalla la estructura del sistema a nivel de entidades, mostrando las clases principales, sus atributos, métodos y relaciones. Esta vista representa elementos como *Cliente*, *Servicio*, *Reserva*, *Producto*, *Reseña* e *Inventario*, definiendo la lógica interna y la organización del modelo de datos.
* **Vista de Procesos (Diagrama de Actividad):** Describe el flujo de actividades y procesos internos del sistema, incluyendo la coordinación entre módulos como el proceso de reserva, el flujo de validación de reseñas, la gestión de inventario y la actualización de información en el panel administrativo. Permite visualizar las dependencias y la secuencia de ejecución entre tareas.
* **Vista de Despliegue (Diagrama de Componentes):** Representa la distribución lógica de los componentes del sistema sobre la infraestructura de software, incluyendo el backend en Node.js, la API REST, el frontend web y la base de datos PostgreSQL. Esta vista refleja cómo se estructuran los módulos y cómo se comunican entre sí.
* **Vista Física (Diagrama Topológico):** Describe la infraestructura de red donde se despliega el sistema, mostrando la relación entre el servidor en la nube (Render), la base de datos remota, la capa de cliente (navegador web) y los servicios externos utilizados. Permite comprender la topología física y los canales de comunicación del sistema.

## Framework Conceptual

Descripción de la arquitectura en contexto

El sistema *Cosmetología Morín* fue diseñado y desarrollado bajo un enfoque ágil, utilizando una **metodología híbrida** que combina prácticas de Scrum —como el uso de sprints, backlog y reuniones de revisión— con elementos tradicionales de documentación formal. Este enfoque permitió realizar entregas incrementales, recibir retroalimentación constante de la emprendedora y adaptarse a cambios en los requisitos durante el ciclo de vida del proyecto, manteniendo a la vez una estructura clara de planificación y control.

La arquitectura se organiza utilizando el Modelo 4+1 de Kruchten, el cual proporciona múltiples perspectivas que permiten una comprensión integral del sistema. Cada vista aborda un aspecto distinto del diseño y su interacción con los requisitos del negocio:

* **Vista de Escenarios:** Representada mediante diagramas de Casos de Uso que modelan las interacciones de los actores principales —administradora, cliente y trabajador— con el sistema. Incluye flujos como gestionar reservas, administrar servicios, validar reseñas y visualizar dashboards.
* **Vista Lógica:** Compuesta por el Modelo de Clases y el Modelo de Datos. Representa las entidades principales del sistema (Reserva, Cliente, Producto, Servicio, Reseña, Inventario) y las relaciones entre ellas, detallando la estructura conceptual del backend y la base de datos PostgreSQL.
* **Vista de Procesos:** Representada con diagramas de actividad que ilustran los flujos internos del sistema, tales como:
* Proceso de creación y confirmación de reservas.
* Gestión de sesiones y pagos.
* Flujo de validación de reseñas.
* Actualización de inventario y servicios.
* Generación de dashboards y estadísticas.

Estos diagramas permiten visualizar la coordinación entre módulos y la secuencia lógica de operaciones.

**- Vista de Despliegue:** Implementada mediante diagramas de componentes y modelos por capas (MVC). Describe la distribución de los módulos del sistema, que incluyen:

* Backend Node JS.
* API REST.
* Frontend web.
* Base de datos PostgreSQL en la nube (Render).

Refleja cómo se organiza la lógica de negocio, la interfaz y el acceso a datos.

**- Vista Física:** Representa la infraestructura de red utilizada para alojar el sistema, destacando:

* servidor en Render donde se ejecuta la API,
* base de datos PostgreSQL alojada remotamente,
* clientes conectados desde navegadores,
* canales de comunicación seguros mediante SSL.

Esta vista permite comprender cómo viaja la información y cómo se distribuye el sistema en un entorno real. Todas estas vistas están interconectadas, garantizando coherencia y alineación con los requisitos funcionales y no funcionales del proyecto. Ninguna vista existe de manera aislada; cada una complementa a las demás, contribuyendo a una visión holística de la arquitectura.  
Los diagramas fueron desarrollados utilizando el estándar **UML 2.0**, implementados con herramientas como **StarUML, Lucidchart, Figma y Canva**, facilitando una representación clara y visual de los elementos críticos del sistema.

La arquitectura del sistema se basa en una combinación de:

* **Arquitectura por capas (Modelo MVC):** separando presentación, lógica de negocio y acceso a datos.
* **Patrones orientados a servicios (REST):** permitiendo que el frontend interactúe con el backend mediante endpoints desacoplados.

Cada módulo del sistema está diseñado para dar soporte directo a un conjunto de casos de uso, entre los que destacan:

* **Gestión de reservas:** creación, confirmación y administración del calendario de servicios.
* **Gestión de productos y servicios:** control de catálogo, precios, descripciones e inventario.
* **Gestión de clientes y sesiones:** historial de servicios, pagos e información relevante.
* **Módulo de reseñas:** validación y publicación de opiniones.
* **Panel administrativo:** herramientas de gestión global para la administradora.
* **Dashboards analíticos:** visualización de ventas, ingresos y servicios más solicitados.

## Actores Usuarios y sus roles

Este documento representa la identificación de Actores/Usuarios Stakeholders y sus roles a partir de la interpretación de los casos de uso del Negocio asociados.

|  |  |  |
| --- | --- | --- |
| ACTOR | ROL / CARGO | FUNCION |
| Administradora | Dueña del negocio / Gestora principal | Gestiona el catálogo de productos y servicios, administra reservas (crear, confirmar, modificar o eliminar), visualiza y edita el historial completo de clientes, gestiona inventario, reseñas y sesiones. Accede a dashboards con indicadores del negocio y tiene control total del panel administrativo y de los usuarios trabajadores. |
| Trabajador | Profesional del área (Distintas áreas) | Visualiza únicamente las reservas asociadas a su área y desde la fecha actual en adelante. Puede marcar sesiones como pagadas (pago final), añadir información a la ficha del cliente, ver el historial de clientes de su área, modificar su contraseña y visualizar sus propios datos. No tiene acceso a módulos de administración global. |
| Cliente | Usuario externo | Realiza reservas a través del formulario de solicitud, entregando información de contacto y seleccionando servicios. No requiere registro ni autenticación en el sistema. Su participación se limita a iniciar procesos de reserva y posteriormente asistir a la sesión. |

## Actividades de arquitectura en el ciclo de vida

El desarrollo de la arquitectura del sistema *Cosmetología Morín* se abordó utilizando un ciclo de vida **iterativo e incremental**, fundamentado en una **metodología híbrida** que combina prácticas de Scrum con elementos tradicionales de planificación y documentación. Esta combinación permitió mantener entregas continuas de software funcional, mientras se aseguraba una documentación clara y estructurada durante todo el proceso.

La elección de esta metodología respondió a la necesidad de validar tempranamente los requisitos con la emprendedora, realizar ajustes oportunos ante cambios y asegurar que la arquitectura evolucionara en coherencia con los casos de uso y las necesidades del negocio.

El ciclo de desarrollo se organizó en **sprints de 2 a 3 semanas**, donde cada iteración se centró en la construcción o mejora de un módulo específico del sistema. Algunos de los módulos desarrollados por sprint fueron:

* **Sprint 1:** diseño del modelo de datos y estructura base del backend.
* **Sprint 2:** módulo de reservas y agenda interactiva.
* **Sprint 3:** gestión de productos y servicios.
* **Sprint 4:** módulo de clientes, sesiones y pagos.
* **Sprint 5:** panel administrativo completo.
* **Sprint 6:** dashboards y métricas analíticas.
* **Sprint 7:** integración general, pruebas y despliegue en Render.

Al finalizar cada sprint, se realizaba una revisión de los resultados junto con la emprendedora, lo que permitía validar la solución implementada, detectar posibles mejoras, ajustar requerimientos y priorizar las tareas para la siguiente iteración. Este proceso garantizó que la arquitectura evolucionara de manera ordenada, flexible y alineada a los objetivos del proyecto.

## Descripciones prácticas de arquitectura

Durante el proceso de desarrollo, la arquitectura del sistema fue evolucionando desde un escenario principal basado en los casos de uso más relevantes hasta conformar un diseño modular y estructurado. Este proceso se llevó a cabo a través de las siguientes actividades:

**Análisis de Requerimientos**:  
Se realizó un levantamiento detallado de los requisitos del sistema junto a la emprendedora, identificando los casos de uso principales del negocio, tales como:

* **Gestionar Reservas**
* **Administrar Productos y Servicios**
* **Gestionar Clientes y Sesiones**
* **Validar y Publicar Reseñas**
* **Visualizar Dashboards con Indicadores del Negocio**
* **Operar el Rol Trabajador** (historial del área, reservas futuras, pagos finales)

Este análisis permitió definir el alcance del sistema y establecer una arquitectura flexible orientada a procesos.

**Definición de la Vista Lógica:**A partir de los casos de uso, se elaboró el **diagrama de clases UML**, identificando las entidades centrales del sistema y sus relaciones:

* Cliente
* Reserva
* Servicio
* Producto
* Sesión
* Pago
* Reseña
* Usuario (Administrador / Trabajador)

Esta estructura sentó las bases del diseño de la base de datos y la lógica del backend en Node.js.

**Modelo de Datos:**  
Se desarrolló un **MER (Modelo Entidad–Relación)** y un diagrama de base de datos en PostgreSQL, manteniendo coherencia con la vista lógica.  
El modelo incluye relaciones clave como:

* Un cliente puede tener múltiples reservas y sesiones
* Los servicios se vinculan a reservas y a áreas de trabajo
* Las reseñas se asocian a clientes y servicios
* Los trabajadores tienen acceso a datos filtrados según su área

El diseño favorece la escalabilidad y el mantenimiento del sistema.

**Vista de Procesos:**  
Se modelaron los flujos principales mediante **diagramas de actividad**, incluyendo:

* Flujo de creación y confirmación de reservas
* Proceso de registro de sesiones y pagos
* Validación y publicación de reseñas
* Actualización de inventario y catálogo
* Generación de dashboards y métricas

Estos flujos permiten comprender el funcionamiento interno del sistema y coordinar la interacción entre los distintos módulos.

**Vista de Despliegue:**  
Se definió la infraestructura necesaria para alojar el sistema en la nube. El despliegue consiste en:

* **Backend Node.js** alojado en **Render**
* **Base de datos PostgreSQL** en Render con acceso seguro
* **Frontend web** consumiendo la API REST
* Comunicación encriptada mediante **HTTPS (SSL)**

Esta configuración garantiza disponibilidad, escalabilidad y seguridad para el uso diario del sistema.

**Definición de Planes de Pruebas y Técnicas de Testing:**  
Se establecieron los planes de prueba para asegurar la estabilidad y calidad del sistema, incluyendo:

* **Pruebas funcionales** (validación de casos de uso)
* **Pruebas de integración** entre frontend, backend y base de datos
* **Pruebas de usabilidad** con la administradora del negocio
* **Pruebas de regresión** al finalizar cada sprint
* **Corrección iterativa de errores** durante el ciclo de desarrollo

Estas actividades permitieron validar el comportamiento del sistema y garantizar que la arquitectura cumpliera con los requisitos funcionales y no funcionales definidos.

## Documentación de la arquitectura

ISO/IEC 25010: Este estándar fue utilizado como base para evaluar la calidad del sistema en aspectos como **funcionalidad, seguridad, usabilidad, eficiencia, mantenibilidad y fiabilidad**. Cada decisión arquitectónica y cada módulo del sistema se revisó considerando estas métricas, asegurando que la plataforma cumpliera con atributos de calidad esenciales para su operación continua en un entorno real.

IEEE 1471-2000: Este estándar proporcionó la guía para la correcta representación de las **vistas arquitectónicas**, la definición de los elementos del sistema y la identificación de los **stakeholders** involucrados. Permitió organizar las descripciones dentro de la arquitectura utilizando un enfoque formal, estructurado y reconocible por las buenas prácticas de ingeniería de software.

Modelo 4+1 de Kruchten

La arquitectura del sistema se estructuró siguiendo el Modelo 4+1, organizando la documentación en cinco vistas complementarias:

* **Vista Lógica** (clases y entidades del sistema)
* **Vista de Procesos** (flujos internos y actividades)
* **Vista de Desarrollo** (componentes y organización del software)
* **Vista Física** (infraestructura y topología del sistema)
* **Vista de Escenarios** (casos de uso principales del negocio)

Este enfoque permitió visualizar el sistema desde múltiples perspectivas, garantizando coherencia, trazabilidad y alineación con los requisitos del proyecto *Cosmetología Morín*.

## Planes de prueba

Durante el desarrollo del sistema se definieron y aplicaron diversos tipos de pruebas para garantizar el correcto funcionamiento de la arquitectura y la calidad del producto final. Entre ellas se incluyen:

**1. Pruebas Funcionales:**  
Validan que cada caso de uso cumpla su propósito:

* Crear y confirmar reservas
* Administrar productos y servicios
* Actualizar sesiones y pagos
* Operar el rol trabajador
* Visualizar dashboards e indicadores

**2. Pruebas de Integración:**  
Aseguran que los módulos trabajen correctamente entre sí:

* Comunicación entre frontend y backend
* Interacción API REST ↔ PostgreSQL
* Flujo completo reserva → sesión → pago

**3. Pruebas de Usabilidad:**  
Realizadas con la emprendedora para evaluar:

* Facilidad de navegación
* Claridad de la interfaz
* Experiencia del rol trabajador
* Fluidez del proceso de reserva

**4. Pruebas de Seguridad:**  
Incluyeron validaciones como:

* Manejo seguro de contraseñas
* Acceso restringido por rol
* Uso de HTTPS y SSL
* Protección de datos sensibles

**5. Pruebas de Regresión:**  
Aplicadas al finalizar cada sprint para asegurar que las mejoras o nuevas funcionalidades no generarán errores en los módulos previamente construidos.

**6. Pruebas de Despliegue:**  
Validaron el correcto funcionamiento del sistema en el entorno productivo (Render):

* Disponibilidad de la API
* Conexión estable con PostgreSQL
* Rendimiento y tiempos de respuesta

## Selección de puntos de vista de la arquitectura

|  |  |  |
| --- | --- | --- |
| VISTAS | UML | Cantidad |
| Escenario Principal | Diagrama de Caso de uso | 1 |
| Vista Lógica | Diagrama de Clases  MER | 1  1 |
| Vista de Proceso | Diagrama de Actividad:  Flujo de reserva | 1 |
| Vista Física | Diagrama Topológico de Servicios | 1 |
| Vista de Despliegue | Diagrama de componentes  Diagrama de Capas (MVC) | 1  1 |

## Vistas de la arquitectura

|  |
| --- |
| UML-VISTA DE ESCENARIO |
| Diagrama de Caso de uso general |
|  |

* **Gestión de Reservas:**  
  Permite al cliente seleccionar un servicio, elegir un horario disponible e ingresar sus datos. El sistema valida automáticamente la disponibilidad y registra la reserva en la agenda. Incluye la confirmación mediante abono inicial, bloqueo del horario y envío de notificaciones al cliente.
* **Gestión de Servicios:**  
  Permite al Administrador crear, modificar y eliminar servicios estéticos, definiendo título, subtítulo, descripción, valor e imagen. Estos servicios pueden estar asociados a áreas de trabajo como cosmetología, masoterapia o estética avanzada.
* **Gestión de Productos:**Permite administrar el catálogo de productos utilizados en la prestación de servicios, incluyendo la actualización de stock, precios y descripciones. Facilita la gestión del inventario de insumos y productos a la venta.
* **Gestión de Usuarios y Trabajadores:**Facilita la creación y administración de cuentas de trabajadores, incluyendo roles, áreas de trabajo, credenciales de acceso y datos personales. Permite organizar al equipo según su especialidad y sus funciones dentro del sistema.
* **Gestión de Clientes:**Permite registrar y administrar clientes, incluyendo nombre, teléfono y RUT. El sistema mantiene un historial de reservas y permite consultar la ficha clínica asociada a cada cliente.
* **Notificación de Reservas y Recordatorios:**El sistema envía automáticamente notificaciones al cliente con la confirmación de su reserva, avisos de cambios en el horario o cancelaciones por falta de abono. Garantiza que el cliente esté al tanto del estado de su atención.
* **Visualización de Agenda:**Los trabajadores pueden consultar su agenda personal, visualizar detalles de sus próximas atenciones y revisar historial de reservas asignadas. Esto permite organizar el flujo de trabajo diario.
* **Registro de Pagos:**Permite registrar abonos y pagos finales asociados a una reserva. El sistema almacena el tipo de pago, método utilizado, fecha y monto, generando historial financiero para la administración.
* **Gestión de Fichas Clínicas:**  
  Los trabajadores pueden crear y completar fichas clínicas para cada atención, registrando observaciones, detalles del procedimiento y productos utilizados. Permite mantener información histórica del tratamiento de cada cliente.
* **Gestión de Reseñas:**  
  Los clientes pueden enviar reseñas sobre su experiencia. El administrador puede aprobar, ocultar o eliminar reseñas antes de publicarlas para mantener control de la reputación digital.
* **Consulta de Catálogo de Servicios y Productos:**  
  Los clientes pueden visualizar el catálogo actualizado de servicios disponibles, imágenes, precios y descripciones. También pueden consultar productos asociados a tratamientos.
* **Envío de Consultas:**  
  El cliente puede enviar consultas desde el sistema, las cuales son recibidas por el administrador para dar seguimiento y responder oportunamente.
* **Administración General del Sistema:**El administrador supervisa módulos completos como agenda, pagos, servicios, productos, fichas clínicas y reseñas, permitiendo una gestión centralizada del funcionamiento del centro de estética.

|  |
| --- |
| UML-VISTA LÓGICA-DIAGRAMA DE CLASES |
| [Diagrama de Clases] |
| mermaid-diagram-2025-11-10-214803.png |

El **Diagrama de Clases** para el sistema de gestión de condominios muestra las clases involucradas, sus atributos, métodos y las relaciones entre ellas. Este diagrama ayuda a modelar la estructura y el comportamiento del sistema de una manera orientada a objetos.

# Elementos Principales

Clases:

* Usuario  
  **Atributos:** ID, username, password\_hash, nombre\_completo, correo, ID\_Area.  
   **Métodos:** crearCuenta(), iniciarSesion().

Área de Trabajo:

* **Atributos:** ID\_Area, nombre\_area, created\_at.  
   **Métodos:** asociarUsuario(), listarServicios().

Cliente:

* **Atributos:** ID\_Cliente, nombre\_cliente, rut\_cliente, telefono\_cliente, fecha\_registro.  
   **Métodos:** actualizarDatos(), consultarHistorial().

Servicio:

* **Atributos:** ID\_Servicio, titulo, subtitulo, descripcion, valor, imagen\_url, fecha\_creacion, ID\_Area.  
   **Métodos:** crearServicio(), modificarServicio(), eliminarServicio().

Producto:

* **Atributos:** ID\_Producto, nombre, descripcion, valor, stock, imagen\_url, created\_at.  
   **Métodos:** actualizarStock(), modificarProducto(), eliminarProducto().

Reserva:

* **Atributos:** ID\_Reserva, fecha\_reserva, hora\_reserva, fecha\_creacion, ID\_Servicio, estado\_reserva, ID\_Cliente.  
   **Métodos:** confirmarReserva(), cancelarReserva().

Pago:

* **Atributos:** ID\_Pago, ID\_Reserva, tipo\_pago, monto\_pagado, metodo\_pago, fecha\_pago, registrado\_por, fecha\_creacion.  
   **Métodos:** registrarPago(), generarComprobante().

Ficha Clínica:

* **Atributos:** ID\_Ficha, ID\_Reserva, detalle, registrado\_por.  
   **Métodos:** registrarFicha(), consultarFicha().

Reseña:

* **Atributos:** ID\_Resena, comentario, fecha\_creacion, estado\_aprobacion, nombre, calificacion.  
   **Métodos:** aprobarResena(), rechazarResena(), publicarResena().

## Relaciones

* **Usuario** tiene una relación de **asociación** con **Condominio** (múltiples condominios).
* **Condominio** tiene una relación de **composición** con **Propiedad** y **Espacio Común** (un condominio contiene propiedades y espacios).
* **Propiedad** tiene una relación de **asociación** con **Gasto Común** (múltiples gastos asociados).
* **Gasto Común** tiene una relación de **asociación** con **Pago** y **Multa**.
* **Espacio Común** tiene una relación de **asociación** con **Reserva de Espacio** (múltiples reservas pueden existir para un espacio común).
* **Historial de Actividades** tiene una relación de **asociación** con **Usuario** (registrar acciones del usuario)

|  |
| --- |
| UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS |
| [Diagrama de BD - MER] |
|  |

El **Modelo Entidad-Relación (MER)** para el sistema de gestión de condominios representa las entidades involucradas, sus atributos y las relaciones que existen entre ellas. Este modelo permite visualizar cómo se organiza la información relacionada con la administración del condominio.

## Elementos Principales

* **Entidades**:
  + **Usuario**:
    - **Atributos**: ID\_Usuario (PK), Nombre, Apellido, Rol, Email, Contraseña, Estado.
  + **Condominio**:
    - **Atributos**: ID\_Condominio (PK), Nombre del Condominio, Dirección, Estado, Tipo.
  + **Propiedad**:
    - **Atributos**: ID\_Propiedad (PK), Número de Propiedad, ID\_Condominio (FK), Tipo de Propiedad.
  + **Gasto Común**:
    - **Atributos**: ID\_Gasto\_Común (PK), Monto Total, Descripción, Mes/Año de Facturación, Estado de Pago, ID\_Propiedad (FK), ID\_Administrador (FK).
  + **Multa**:
    - **Atributos**: ID\_Multa (PK), Monto, Motivo, Fecha de Generación, Fecha de Vencimiento, Estado, ID\_Gasto\_Común (FK), ID\_Propiedad (FK).
  + **Pago**:
    - **Atributos**: ID\_Pago (PK), Monto Pagado, Fecha de Pago, ID\_Gasto\_Común (FK), ID\_Usuario (FK), Método de Pago.
  + **Espacio Común**:
    - **Atributos**: ID\_Espacio\_Común (PK), Nombre del Espacio, Capacidad, Costo de Reserva, Estado, ID\_Condominio (FK).
  + **Reserva de Espacio**:
    - **Atributos**: ID\_Reserva (PK), Fecha y Hora de Inicio, Fecha y Hora de Fin, ID\_Espacio\_Común (FK), ID\_Usuario (FK), Estado.
  + **Historial de Actividades**:
    - **Atributos**: ID\_Actividad (PK), Tipo de Actividad, Descripción, ID\_Usuario (FK), Fecha y Hora de la Actividad.

## Relaciones

Cliente tiene una relación de asociación con Reserva (un cliente puede realizar múltiples reservas dentro del sistema).

Servicio tiene una relación de asociación con Reserva (un servicio puede estar asociado a múltiples reservas realizadas por distintos clientes).

Reserva tiene una relación de asociación con Pago (una reserva puede generar uno o varios pagos asociados, dependiendo del tipo de abono o pago final).

Reserva tiene una relación de composición con Ficha Clínica (una ficha clínica pertenece estrictamente a una única reserva; si la reserva se elimina, la ficha también desaparece).

Usuario tiene una relación de asociación con Pago y Ficha Clínica (un usuario —trabajador— puede registrar múltiples pagos y múltiples fichas clínicas en el sistema).

Área de Trabajo tiene una relación de composición con Servicio (cada área agrupa los servicios asociados; si un área deja de existir, los servicios vinculados dejan de estar activos).

Área de Trabajo tiene una relación de asociación con Usuario (un área puede tener múltiples usuarios/trabajadores asignados dependiendo de su rol o especialidad).

|  |
| --- |
| UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD |
| [Diagramas de Actividad] |
|  |

El diagrama de actividad representa el flujo de actividades del proceso de **realizar una reserva de servicio en Cosmetología Morín**. Este diagrama describe las decisiones del cliente, la validación de disponibilidad y la intervención del administrador para confirmar o cancelar la reserva según el abono recibido.

## Componentes del Proceso

**Inicio:** El cliente accede al sistema para agendar un servicio estético.

**Seleccionar Servicio:** El cliente selecciona un servicio desde el catálogo disponible (por ejemplo, limpieza facial, masoterapia, tratamientos estéticos).

**Seleccionar Horario:** El cliente visualiza los horarios disponibles para el servicio elegido.

* **Si un horario está ocupado:** el sistema lo muestra como **“Reservado”** y no puede seleccionarse.
* **Si el horario está disponible:** el cliente puede continuar al siguiente paso.

**Ingresar Datos:** El cliente ingresa la información requerida (nombre, teléfono y otros datos relevantes).

**Registrar Solicitud en Agenda:** El sistema registra la reserva como *pendiente*, a la espera del abono correspondiente.

**Validación del Abono Inicial:** El administrador revisa si el cliente realizó el abono para confirmar la reserva.

Si NO existe abono:

* El administrador cancela la reserva.
* El horario se libera nuevamente en la agenda.

Si SÍ existe abono:

* El administrador confirma la reserva.
* El horario queda bloqueado definitivamente.

**Notificación al Cliente:** El **administrador** envía una notificación al cliente indicando el resultado del proceso:

* Reserva confirmada
* Reserva cancelada por falta de abono.

|  |
| --- |
| UML-VISTA DE DESPLIEGUE |
| Sistema de Gestión de Condominios |
|  |

# Servidor de Aplicaciones

* **Aplicación Web de Gestión y Reservas:** Corresponde al backend desarrollado en Node.js y desplegado en Render.  
  Proporciona los servicios esenciales del sistema, incluyendo la gestión de reservas, servicios, clientes, pagos y fichas clínicas.  
  Actúa como intermediario entre el frontend y la base de datos, procesando peticiones, validaciones y actualizaciones de información.

# Servidor de Base de Datos

* **Base de Datos Cosmetología Morín:** Almacenada en Supabase (PostgreSQL), guarda toda la información relevante del sistema:  
  clientes, usuarios, trabajadores, servicios, productos, reservas, pagos, fichas clínicas y reseñas.  
  Este servidor es fundamental para el funcionamiento de la plataforma, ya que responde a consultas, ejecuta transacciones y garantiza la persistencia e integridad de los datos.

# Servidor de Notificaciones (Operación Manual)

* **Sistema de Notificación Administrativa:** A diferencia de otros sistemas automatizados, las notificaciones en Cosmetología Morín son enviadas **manualmente por el administrador**, quien informa al cliente sobre:  
  confirmación de reservas, validación de abonos, cambios de horario o cancelaciones.  
  Este proceso permite mantener un contacto directo con el cliente, asegurando una comunicación personalizada acorde al estilo del negocio.

# Navegación de Clientes y Trabajadores (Frontend Web)

* **Interfaz Web del Sistema:** Accedida por clientes, trabajadores y el administrador.  
  Permite visualizar servicios, realizar reservas, registrar atenciones, administrar productos y gestionar la operación completa del centro estético.

|  |
| --- |
| UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA |
| Sistema Gestión De Condominios |
|  |

## Centro de Datos Principal

Es el corazón del sistema donde se aloja la infraestructura central en la nube. Incluye los servidores necesarios para la operación de la plataforma web, tales como:

* **Servidor de Aplicaciones (Render):** contiene la API RESTful desarrollada en Node.js, encargada de procesar reservas, pagos, servicios, fichas clínicas y usuarios.
* **Servidor de Base de Datos (Supabase – PostgreSQL):** almacena toda la información del sistema, incluyendo clientes, trabajadores, reservas, servicios y pagos.
* **Firewall y Seguridad Cloud:** mecanismos de protección nativos de Render y Supabase para garantizar la comunicación segura y el acceso autenticado.

Estos componentes permiten un sistema escalable y disponible sin requerir infraestructura física local.

## Red Local del Centro Cosmetología Morín

Esta red incluye los dispositivos de uso interno en el negocio, tales como:

* **Estación de trabajo del Administrador**, utilizada para gestionar reservas, validar abonos, administrar servicios y enviar notificaciones al cliente.
* **Estación de trabajo del Trabajador**, utilizada para revisar la agenda diaria, consultar clientes y registrar fichas clínicas.

La red local se conecta al Centro de Datos mediante el router del negocio, permitiendo el acceso al sistema alojado en la nube.

## Conexiones Externas

El acceso a Internet se proporciona a través de un **Proveedor de Servicios de Internet (ISP)**, permitiendo que:

* El administrador y los trabajadores accedan al sistema web.
* Los servicios de Render y Supabase estén disponibles para los usuarios internos.

El ISP actúa como puente entre la red local y la infraestructura alojada en la nube.

## Dispositivos de los Clientes

Aunque no se muestran explícitamente en este diagrama, los clientes pueden acceder utilizando sus dispositivos personales —teléfonos móviles, tablets o computadores— para:

* Agendar servicios
* Visualizar el catálogo
* Enviar consultas
* Crear reseñas

Todo el acceso se realiza directamente a través del **Servidor de Aplicaciones en Render**.

|  |
| --- |
| Usabilidad y Calidad de software- Front-end -Vistas Principales |
| Vistas principales del sistema – Vista Home Principal de entrada |
|  |

El diseño del frontend se desarrolló con un estilo **moderno, minimalista y coherente con la identidad de Cosmetología Morín**.  
Incluye secciones dinámicas de servicios y productos, un menú tipo hamburguesa, un botón flotante para volver al inicio y formularios con validaciones visuales.  
Se presenta un diseño limpio y ordenado que resalta las opciones más importantes, permitiendo a los usuarios interactuar con el sistema de manera eficiente.

|  |
| --- |
| Usabilidad y Calidad de software – Vistas específicas de usuario |
| Vistas principales del sistema – Vista Principal de Usuario a escritorio de trabajo principal de su componente de sistema |
|  |

* **Administrador:** Gestión completa de reservas, servicios, productos y reseñas, además de acceso a dashboards y administración general del sistema.
* **Trabajador:** Visualización y gestión de las reservas asignadas según su área, actualización de estados y consulta de su historial.
* **Cliente:** Solicitud de servicios, ingreso de reseñas

# Consistencia en la cantidad de vistas de la arquitectura

Se garantiza que cada vista esté alineada con los requisitos funcionales y no funcionales del sistema, evitando redundancias y manteniendo coherencia con los casos de uso, diagramas UML, BPMN y los prototipos del proyecto.

Cada vista corresponde directamente a un rol definido y a los módulos implementados en el backend y frontend.

DESCRIPCION DE MODULOS

|  |  |  |
| --- | --- | --- |
| Nombre del módulo | Descripción | Componentes incluidos |
| Gestión de Reservas | Permite crear, editar, asignar y administrar reservas en todas las áreas. | Motor de reservas, módulo de agenda, módulo de asignación por área, módulo de estados. |
| Gestión de Servicios | Controla la administración de servicios ofrecidos (faciales, pestañas, masajes, etc.). | Catálogo de servicios, módulo de áreas, módulo de edición. |
| Gestión de Productos | Permite administrar productos utilizados o vendidos dentro del centro. | Inventario, módulo CRUD, módulo de categorías. |
| Gestión de Usuarios | Administración de cuentas y roles (cliente, trabajador, administrador). | Módulo de registro, autenticación, perfiles. |
| Reseñas | Módulo de moderación de reseñas enviadas por clientes. | Moderador de reseñas, módulo de visibilidad. |
| Dashboard Administrativo | Muestra métricas clave como demanda, historial, ocupación y actividades. | Módulo de reportes, visualizaciones y KPIs. |

DESCRIPCIÓN DE COMPONENTES

|  |  |  |
| --- | --- | --- |
| Nombre del Componente | Descripción | Componentes Relaciones |
| Motor de Reservas | Gestiona la creación, validación y estado de reservas. | Módulo de Reservas, Módulo de Usuarios. |
| Motor de Notificaciones | Envía notificaciones por correo y mensajes emergentes | Módulo de Reservas, Módulo de Clientes. |
| Gestor de Usuarios | Maneja la creación y eliminación de Usuarios | Autenticación, Módulo de Perfiles. |
| Moderador de Reseñas | Permite mostrar, ocultar o eliminar reseñas. | Módulo de Reseñas. |
| Dashboard | Procesa datos y genera KPIs. | Reservas, Servicios, Clientes. |

DESCRIPCIÓN DE CONECTORES

|  |  |  |
| --- | --- | --- |
| Nombre del Conector | Tipo | Propósito |
| REST API (Express.js) | Comunicación | Transferencia de datos entre frontend y backend. |
| Supabase PostgreSQL SSL | Persistencia | Almacenamiento y consulta de datos. |

**Arquitectura lógica:** La arquitectura lógica del sistema está basada en una estructura cliente-servidor que separa claramente el frontend, backend y la capa de datos. El sistema sigue una arquitectura modular, permitiendo mantener organizadas las funcionalidades por componentes (reservas, servicios, usuarios, reseñas, productos y dashboards).

La comunicación entre frontend y backend se realiza mediante API REST, mientras que la persistencia de datos se gestiona a través de un motor PostgreSQL alojado en la plataforma Supabase. El despliegue del servidor se efectúa en Render, permitiendo integración continua desde GitHub.

**Performances:** El sistema está optimizado para obtener tiempos de respuesta rápidos y un uso eficiente de recursos.  
 Se aplican las siguientes estrategias:

* **Optimización de consultas:** Uso de índices y operaciones asíncronas en PostgreSQL para mejorar el rendimiento.
* **Pool de conexiones:** Manejo eficiente de conexiones entre backend y base de datos.
* **Carga bajo demanda:** Datos que no cambian frecuentemente se cargan solo cuando es necesario.
* **Compresión automática:** Render utiliza compresión para mejorar los tiempos de carga.

**Criterios de Calidad:** El sistema cumple con los siguientes criterios:

* **Disponibilidad:** Backend desplegado en Render y base de datos en Supabase con backups automáticos.
* **Respuesta rápida:** Las transacciones se mantienen por debajo de 2 segundos.
* **Seguridad:** Conexión SSL, hash de contraseñas y control de roles para proteger la información.
* **Escalabilidad:** Arquitectura preparada para crecer sin afectar el rendimiento.

**Detalles de la implementación:** La implementación del sistema se basará en una arquitectura de microservicios, utilizando las siguientes tecnologías:

* **Backend:** Node.js + Express para API REST.
* **Frontend:** HTML, CSS, JavaScript y componentes React.
* **Base de Datos:** PostgreSQL en Supabase con conexión SSL.
* **Infraestructura:** Render para el despliegue y GitHub para integración continua.

**Lenguajes y plataformas:** Las tecnologías específicas que se utilizarán en el desarrollo del sistema incluyen:

* Node.js
* Express
* React
* PostgreSQL (Supabase)
* Render Visual Studio Code
* GitHub

# Acceso a fuentes de información del proyecto

Para acceder a la carpeta de evidencias acumuladas y otros documentos relevantes del proyecto, puede utilizar el siguiente enlace:

<https://github.com/Pvbaeza/2025_MA_CAPSTONE_704D_GRUPO_1>  
https://github.com/Pvbaeza/cosmetica  
  
Esta carpeta incluye documentación técnica, diagramas, pruebas de rendimiento y cualquier otra evidencia relacionada con el desarrollo y la implementación del sistema