PROYECTO ARQUITECTURA DE LA INFORMACIÓN

DAVID SANTIAGO AVILA RINCÓN

DANIEL CHAVES PARRA

STEVEN ANDRES MADERA MARTINEZ

JUAN CARLOS MONCADA CASTRO

JUAN PABLO VILLA RAMIREZ

UNIVERSIDAD LIBRE

ARQUITECTURA DE LA INFORMACIÓN

2025-1

Contenido

[INTRODUCCIÓN 3](#_Toc196741673)

[OBJETIVOS 4](#_Toc196741674)

[Objetivos Generales: 4](#_Toc196741675)

[Objetivos Específicos: 4](#_Toc196741676)

[DESARROLLO DEL PROYECTO 5](#_Toc196741677)

[Patrones de Diseño 5](#_Toc196741678)

[Singleton: 5](#_Toc196741679)

[Factory Method: 6](#_Toc196741680)

[Observer: 7](#_Toc196741681)

[Microservicios 8](#_Toc196741682)

[Código de interacción: 9](#_Toc196741683)

[Código de interacción: 10](#_Toc196741684)

[Código de interacción: 10](#_Toc196741685)

[Código de interacción: 11](#_Toc196741686)

[Contenedores 12](#_Toc196741687)

[Beneficios de la Contenedorización 12](#_Toc196741688)

[Dokerts y Kubernetes 14](#_Toc196741689)

[Contenedorización (Adaptado a Ristorante Bella Napoli) 14](#_Toc196741690)

[Kubernetes (Despliegue en Cluster para Ristorante Bella Napoli) 15](#_Toc196741691)

[Diseño Responsivo y Móvil 18](#_Toc196741692)

[Análisis de Usuario y Usabilidad 20](#_Toc196741693)

[Estructuración de Contenido 21](#_Toc196741694)

[Site MAP 23](#_Toc196741695)

[Wireflows 25](#_Toc196741696)

[CONCLUSIONES 27](#_Toc196741697)

[BIBLIOGRAFÍA 28](#_Toc196741698)

# 

# INTRODUCCIÓN

En una época digital donde los requerimientos de una reserva se convierten en una oportunidad de solución además la creciente demanda de los usuarios por soluciones rápidas, confiables y accesibles ha impulsado el desarrollo de plataformas tecnológicas que centralizan y simplifican estos procesos.

Este informe describe el planteamiento, el diseño, el desarrollo y la implementación del proyecto, detallando cada una de las etapas, las tecnologías utilizadas y los beneficios esperados tanto para los usuarios como para los prestadores de servicios.

Además, gracias a la información suministrada por parte de la arquitectura de la información para encontrar la mejor manera de que el usuario puede tener un manejo fácil, intuitivo y rápido para cumplir su objetivo.

Este informe permite saber el diseño, las estrategias, la estructuración, patrones, contenedores, kubernetes utilizados para el mejor avance de este proyecto y permitir que surja a través de la excelencia y perfección

# OBJETIVOS

## Objetivos Generales:

1. Desarrollar una plataforma integral de reservas que permita a los usuarios gestionar espacios y servicios como parqueaderos, restaurantes, boletería y bares de manera eficiente y segura.

2. Optimizar los procesos de reserva y administración de espacios en distintos establecimientos, mejorando la experiencia del cliente y la gestión operativa.

## Objetivos Específicos:

1. Diseñar una interfaz de usuario intuitiva que facilite la navegación y la realización de reservas en pocos pasos.

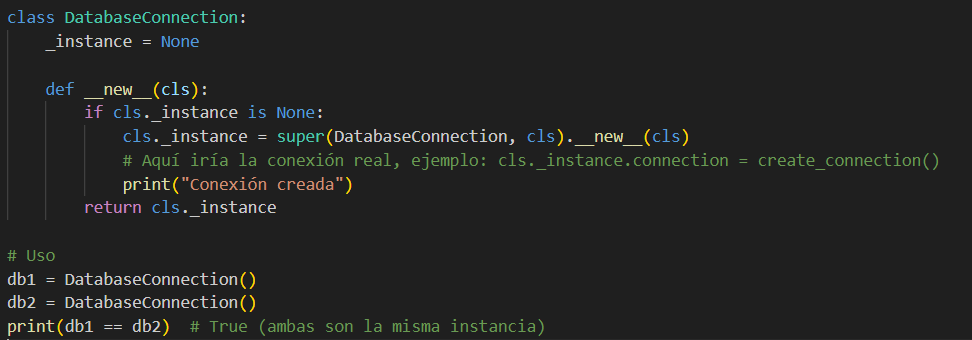
2. Implementar un sistema de notificaciones y confirmaciones automáticas para garantizar la comunicación efectiva entre el usuario y el establecimiento.

3. Integrar métodos de pago seguros que permitan la reserva anticipada y aseguren la disponibilidad de los servicios solicitados.

# DESARROLLO DEL PROYECTO

## Patrones de Diseño

### Singleton:



Singleton fue escogido y usado para controlar recursos que deben ser únicos, como la conexión a la base de datos de las reservas. Se usa este patrón de diseño puesto que el sistema de reservas necesita una única conexión centralizada a la base de datos para manejar reservas, usuarios, pagos, etc. Si cada módulo abriera su propia conexión sería un caos (duplicados, errores, pérdida de datos).

La principal ventaja de Singleton en cuanto al código y su usabilidad es que ahorra memoria y evita problemas de múltiples conexiones a una misma base de datos (cosa que en un canal de reservas como el proyectado sería un caos total).

Su función principal es controlar los recursos para que los mismos sean únicos, por ejemplo, una de las mesas de un restaurante no puede ser para dos clientes diferentes a la misma hora.

### Factory Method:

Texto

El contenido generado por IA puede ser incorrecto.

El patrón de diseño de Factory Method permite delegar la creación de objetos (como reservas de parqueadero, restaurante, bar) a una "fábrica" especializada.

¿Qué hace el patrón de diseño? El patrón de diseño permite crear objetos sin especificar una clase

Se eligió este patrón de diseño puesto que su mas grande ventaja es su flexibilidad a la hora de agregar nuevos tipos de reservas en el futuro sin tocar el código que ya funciona.

Y elegimos este patrón de diseño por una característica especifica que nos permitirá en prototipos posteriores un gran avance, si queremos agregar otro tipo de reserva por un evento o un diferente servicio se puede agregar sin tener la necesidad de reescribir el sistema completo.

### Observer:

Texto

El contenido generado por IA puede ser incorrecto.

El patrón de diseño Observer fue utilizado para automatizar acciones en respuesta de eventos, es decir, notificar al cliente que la reserva está confirmada por medio de un correo electrónico. Se tomó este patrón de diseño puesto que permite que varios objetos (observadores) se enteren automáticamente cuando algo cambia en otro objeto (observable).

Decidimos usar Observer ya que existe una menor dependencia entre módulos y permite una reducción de acoplamientos. Lo que permite que cada módulo escuche el cambio sin la necesidad de preguntar constantemente si un cambio fue hecho. Fue utilizado ya que se necesita que, al confirmarse una reserva, se envíen varias notificaciones (correo o SMS) sin que el sistema principal tenga que preocuparse de cada canal.

## Microservicios

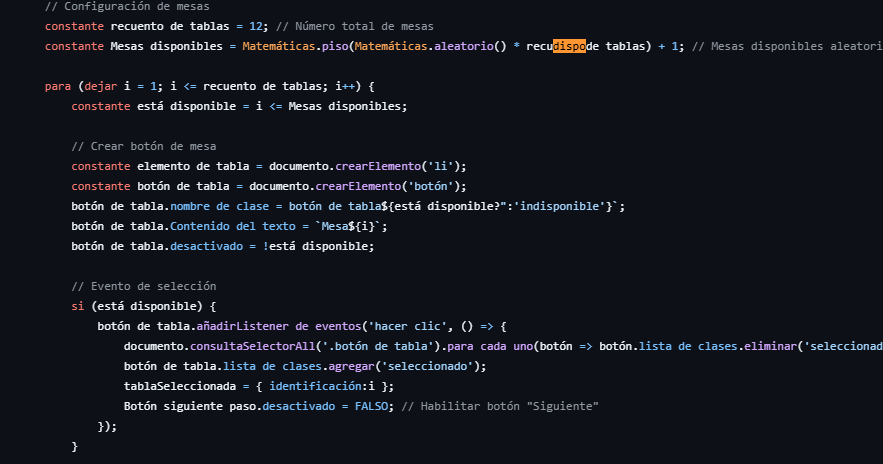
La arquitectura de microservicios ha emergido como un paradigma robusto y flexible para el desarrollo de aplicaciones complejas, ofreciendo una alternativa a las arquitecturas monolíticas tradicionales. En el contexto de un sistema de reservas de restaurante, la adopción de microservicios presenta una serie de ventajas significativas que justifican su implementación y contribuyen a la escalabilidad, mantenibilidad y resiliencia de la plataforma. Basándonos en el análisis del código *frontend* proporcionado y las funcionalidades inherentes a un sistema de reservas, podemos argumentar sólidamente el uso de varios microservicios clave.

Uno de los microservicios fundamentales en este escenario es el Servicio de Disponibilidad de Mesas. La lógica simulada en el *frontend* para la generación aleatoria de horarios y la presentación de mesas disponibles subraya la necesidad de un componente *backend* dedicado a gestionar esta información crítica. Un microservicio de disponibilidad de mesas sería responsable de mantener el inventario de mesas del restaurante, registrar las reservas existentes y determinar en tiempo real qué mesas están disponibles para una fecha y hora específicas.

Su implementación se basa en las siguientes razones:

* Su gestión centralizada de la disponibilidad: Un servicio dedicado asegura una única fuente de verdad para la disponibilidad de mesas, evitando inconsistencias y errores que podrían surgir al dispersar esta lógica en múltiples partes de la aplicación.
* Su escalabilidad independiente: La demanda de consultas de disponibilidad puede variar significativamente. Un microservicio independiente permite escalar este componente de forma individual para manejar picos de tráfico sin necesidad de escalar toda la aplicación.
* La lógica de negocio especializada: La determinación de la disponibilidad puede involucrar reglas de negocio complejas, como la capacidad máxima por mesa, la duración de las reservas y la gestión de mesas especiales. Un microservicio permite encapsular y gestionar estas reglas de manera eficiente.
* Por la integración con otros sistemas: Este servicio podría integrarse con sistemas de gestión de restaurantes (POS) para obtener información en tiempo real sobre el estado de las mesas (ocupadas, limpias, etc.).

### Código de interacción:

****

Otro microservicio esencial es el Servicio de Gestión de Reservas. La acción de confirmar una reserva y la necesidad de almacenar los datos del cliente y los detalles de la reserva (fecha, hora, mesa, número de comensales) apuntan directamente a la necesidad de un componente *backend* encargado de esta tarea.

El uso de este microservicio permite que:

* La persistencia de datos: Un microservicio de gestión de reservas se encargaría de interactuar con una base de datos para almacenar de forma segura y persistente la información de las reservas.
* La validación de reservas: Este servicio podría implementar lógica de validación para asegurar que las reservas cumplan con las reglas del restaurante (por ejemplo, capacidad máxima, horarios de atención).
* El procesamiento de pagos (si aplica): En caso de que el restaurante requiera un depósito o pago por adelantado para las reservas, este servicio podría integrarse con pasarelas de pago.
* La generación de identificadores únicos: Asignar identificadores únicos a cada reserva facilita su seguimiento y gestión.

### Código de interacción:

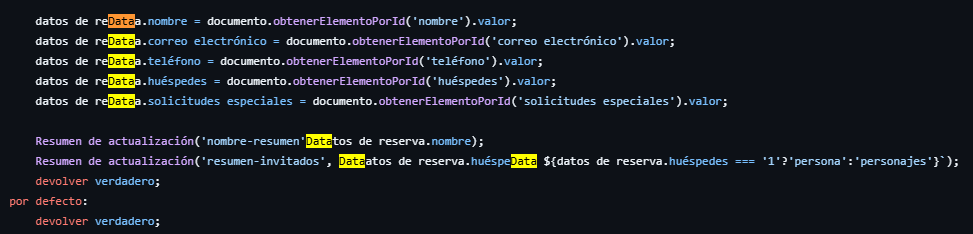
****

Continuando con la argumentación, el Servicio de Autenticación y Usuarios se vuelve crucial si el sistema de reservas permite a los clientes crear cuentas, guardar su información o ver su historial de reservas. Aunque el código *frontend* proporcionado no detalla este aspecto, es una funcionalidad común en sistemas de reservas modernos.

La razones de uso son:

* Seguridad de la información del cliente: Un servicio de autenticación y usuarios gestiona de forma segura la información personal y las credenciales de los clientes.
* Personalización de la experiencia: Al permitir la creación de cuentas, se puede ofrecer una experiencia más personalizada, como recordar las preferencias del cliente o facilitar futuras reservas.
* Gestión de perfiles: Los usuarios pueden gestionar su información personal, ver su historial de reservas y posiblemente cancelar o modificar reservas (sujeto a las políticas del restaurante).

### Código de interacción:

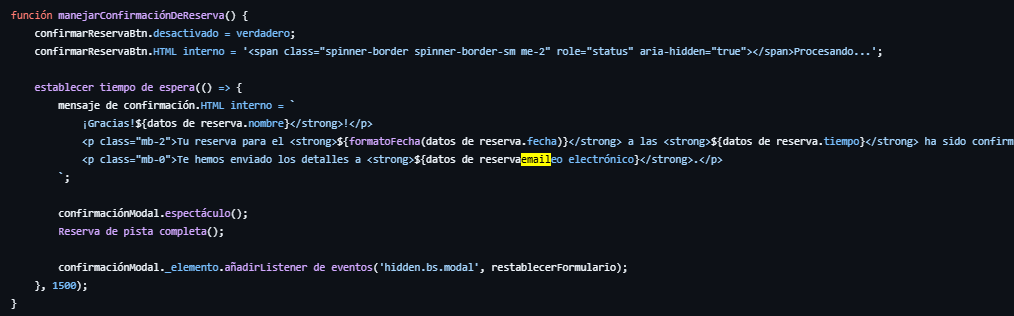


Además, el Servicio de Notificaciones (Email/SMS) es fundamental para mantener a los clientes informados sobre el estado de sus reservas. La mención en el código *frontend* del envío de detalles por correo electrónico tras la confirmación de la reserva resalta esta necesidad.

Junto con los servicios de notificaciones nos permitimos:

* Confirmación de reservas: Enviar una confirmación por correo electrónico o SMS proporciona al cliente un comprobante de su reserva y los detalles importantes.
* Recordatorios de reserva: Enviar recordatorios automáticos reduce la posibilidad de ausencias y mejora la eficiencia del restaurante.
* Notificaciones de cambios o cancelaciones: Informar a los clientes sobre cualquier cambio en su reserva (por ejemplo, un cambio de horario o una cancelación por parte del restaurante) es crucial para una buena experiencia del cliente.

### Código de interacción:

****

La adopción de una arquitectura de microservicios para un sistema de reservas de restaurante ofrece una serie de beneficios tangibles en términos de escalabilidad, mantenibilidad, resiliencia y agilidad en el desarrollo. Los microservicios de disponibilidad de mesas, gestión de reservas, autenticación y usuarios, notificaciones y analíticas son componentes esenciales que permiten construir una plataforma robusta y eficiente, capaz de adaptarse a las necesidades cambiantes del negocio y proporcionar una experiencia de usuario óptima. La separación de funcionalidades en servicios independientes facilita el desarrollo, la prueba y el despliegue individual, contribuyendo a un ciclo de vida del *software* más eficiente y a una mayor capacidad de respuesta ante las demandas del mercado.

## Contenedores

En la arquitectura de software, un contenedor es una unidad de software que empaqueta el código de una aplicación y todas sus dependencias, de modo que la aplicación se ejecute de forma rápida y confiable de un entorno informático a otro. Los contenedores aíslan el software de su entorno y garantizan que funcione de manera uniforme a pesar de las diferencias entre, por ejemplo, los entornos de desarrollo y producción.

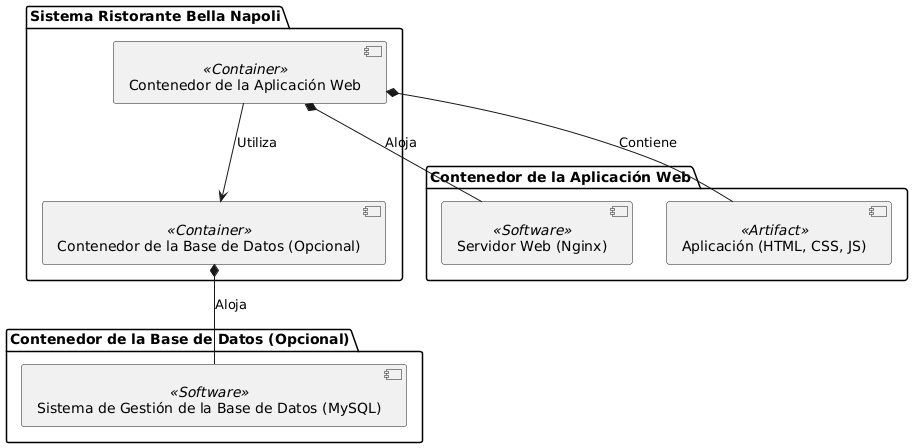
### Beneficios de la Contenedorización

* Portabilidad: La aplicación se puede ejecutar en cualquier sistema que admita contenedores.
* Aislamiento: Los contenedores aíslan las aplicaciones entre sí, evitando interferencias.
* Eficiencia: Los contenedores comparten el kernel del sistema operativo del host, lo que los hace más livianos que las máquinas virtuales.
* Escalabilidad: Los contenedores se pueden escalar fácilmente para manejar una mayor carga.

Aplicación al Código de la Página Web del Ristorante Bella Napoli

Considerando el código HTML proporcionado, una arquitectura de software contenerizada para esta aplicación web podría involucrar los siguientes contenedores:

1. Contenedor de la Aplicación Web:
   * Este contenedor empaqueta el código de la aplicación web (HTML, CSS, JavaScript) junto con un servidor web (como Nginx o Apache).
   * Garantizaría que la aplicación se ejecute de manera consistente en diferentes entornos (desarrollo, pruebas, producción).
   * Facilita la implementación de la aplicación en un servidor o plataforma en la nube.
2. Contenedor de la Base de Datos (Opcional):
   * Si la aplicación web utilizará una base de datos (por ejemplo, para almacenar información de reservas), esta se ejecutaría en su propio contenedor.
   * Este aislamiento mejoraría la seguridad y la mantenibilidad.
   * Permitiría escalar la base de datos independientemente de la aplicación web

****

La contenedorización ofrece una forma poderosa de empaquetar e implementar aplicaciones web como la del Ristorante Bella Napoli. Al utilizar contenedores, podemos garantizar la portabilidad, el aislamiento, la eficiencia y la escalabilidad de la aplicación, lo que lleva a un proceso de desarrollo e implementación más fluido y una arquitectura de software más robusta.

## Dokerts y Kubernetes

Con el crecimiento de los servicios digitales, las aplicaciones modernas requieren ser altamente disponibles, escalables y portables. La página de reservas del restaurante *Bella Napoli* debe responder a estas exigencias.  
 Para lograrlo, se propone la contenerización de la aplicación usando Docker y su despliegue en un clúster de Kubernetes.

Esta estrategia permitirá:

* Escalar la aplicación fácilmente según la demanda.
* Simplificar la gestión y actualización del sistema.
* Asegurar que la aplicación se comporte de forma consistente en cualquier entorno.

### Contenedorización (Adaptado a Ristorante Bella Napoli)

Para facilitar el despliegue y asegurar la consistencia del entorno de desarrollo y producción de la plataforma de reservas del Ristorante Bella Napoli, se implementó Docker.  
 Cada componente clave, como el servidor web , la base de datos (MySQL/PostgreSQL) y la aplicación backend (Node.jsr), fue encapsulado en contenedores separados.  
 Esto permitió aislar dependencias, mejorar la portabilidad del sistema y facilitar la replicación exacta del entorno de producción en las estaciones de los desarrolladores.  
 El uso de Docker Compose permite orquestar todos los servicios esenciales (API de reservas, servicio de notificaciones, motor de autenticación) con un solo comando (docker-compose up), reduciendo tiempos de configuración y minimizando errores derivados de diferencias entre sistemas operativos.

Entre los beneficios principales destacan:

* **Despliegue rápido** de nuevas versiones del sistema de reservas.
* **Facilidad para escalar** servicios específicos como el módulo de gestión de reservas en fechas de alta demanda (San Valentín, Navidad, etc.).
* **Mayor control** sobre versiones de dependencias críticas (bases de datos, lenguajes de programación).



### Kubernetes (Despliegue en Cluster para Ristorante Bella Napoli)

Pensando en un crecimiento futuro y en eventos de alta concurrencia de usuarios (reservas simultáneas en días especiales o promociones), se consideró la implementación de **Kubernetes** como sistema de orquestación de contenedores.  
 Aunque la plataforma del **Ristorante Bella Napoli** no necesita inicialmente una infraestructura de alta escala, se diseñó el sistema de tal forma que pueda ser desplegado en un **clúster de Kubernetes** cuando sea necesario.

Principales ventajas proyectadas:

* **Escalado automático:** Por ejemplo, aumentar dinámicamente las réplicas del microservicio de gestión de reservas cuando la carga de usuarios lo requiera (escenario típico en lanzamientos de menús especiales).
* **Alta disponibilidad:** Kubernetes permite mantener el servicio operativo incluso si fallan algunos nodos del clúster.
* **Actualizaciones sin tiempo de inactividad (zero downtime):** Ideal para mantener la plataforma accesible mientras se lanzan nuevas funcionalidades (como reserva de mesas específicas o preorden de menús).



## Diseño Responsivo y Móvil



La página de reservas del Ristorante Bella Napoli fue diseñada siguiendo un enfoque Mobile-First, asegurando que la experiencia en dispositivos móviles fuera no solo accesible, sino también óptima desde el primer momento. Se implementaron técnicas de media queries estratégicas para adaptar fluidamente la interfaz a diferentes tamaños de pantalla, estableciendo puntos de quiebre específicos para smartphones, tablets y escritorios de alta resolución.

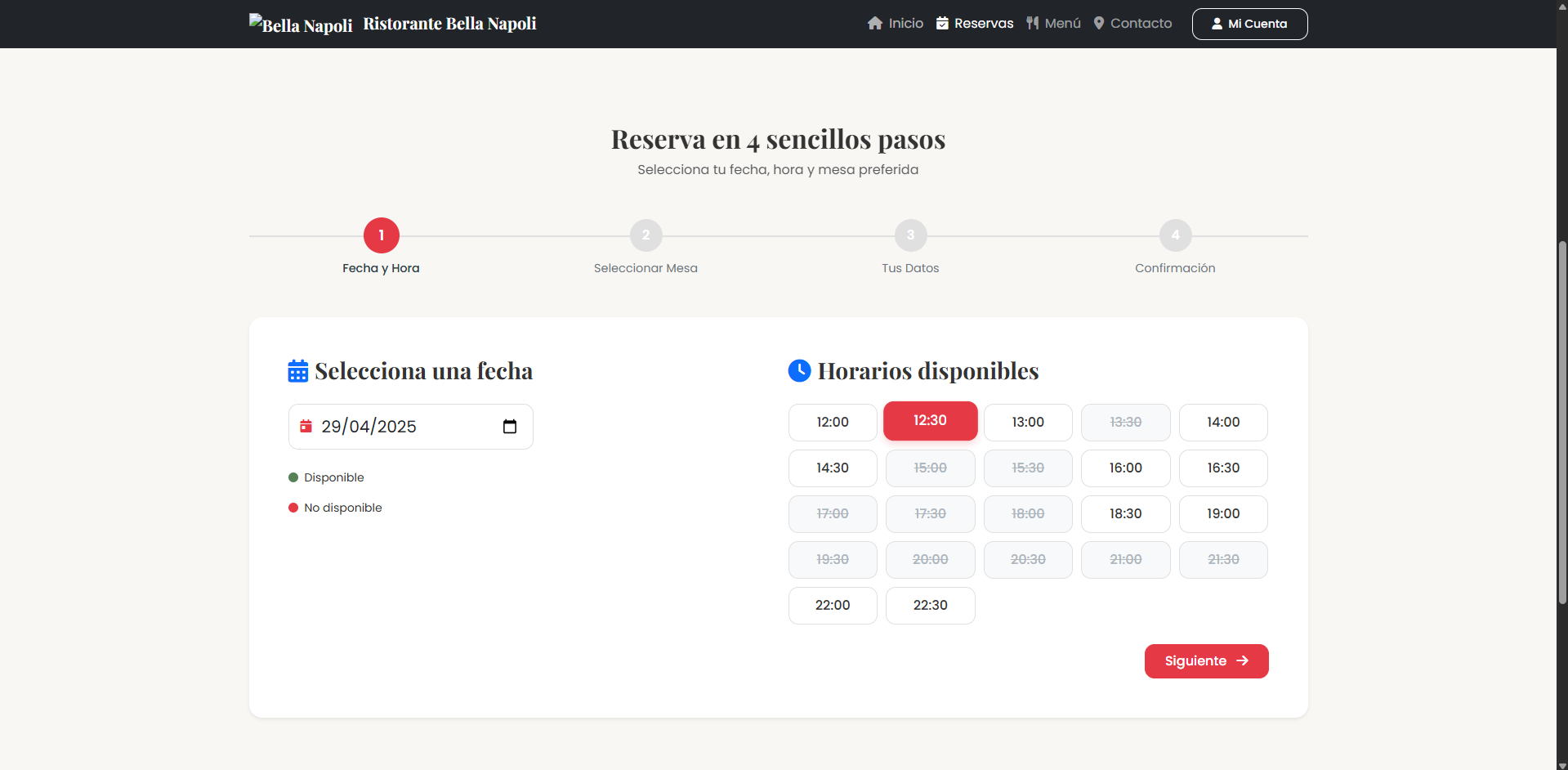
Para potenciar la adaptabilidad y la velocidad de desarrollo, se incorporó el framework Bootstrap 5.3, que permitió construir una arquitectura de componentes flexibles, accesibles y estilizados de manera coherente con los principios de diseño responsivo. Además, se integraron fuentes de Google Fonts y animaciones AOS para enriquecer la experiencia visual sin comprometer el rendimiento en dispositivos con conectividad limitada.

Se optimizaron imágenes (logotipos y gráficos de reserva) y se implementaron buenas prácticas de carga eficiente, incluyendo la minimización de hojas de estilo y scripts como los de animaciones (GSAP) y control de interacciones dinámicas. Esta estrategia contribuyó a reducir los tiempos de carga, fundamental para retener usuarios en entornos móviles, donde la velocidad de navegación es crítica.

La experiencia de reserva fue diseñada en cuatro pasos interactivos, utilizando un sistema de stepper dinámico que guía al usuario de manera intuitiva, maximizando la conversión y la satisfacción. Complementariamente, se integraron sistemas de validación de formularios en tiempo real para mejorar la precisión de los datos ingresados, lo que refuerza la usabilidad y la eficiencia del proceso de reserva.

Finalmente, se incorporaron medidas para mejorar la accesibilidad, como íconos claros (FontAwesome) y etiquetas ARIA implícitas mediante Bootstrap, garantizando que la plataforma sea inclusiva para un público más amplio.

## Análisis de Usuario y Usabilidad

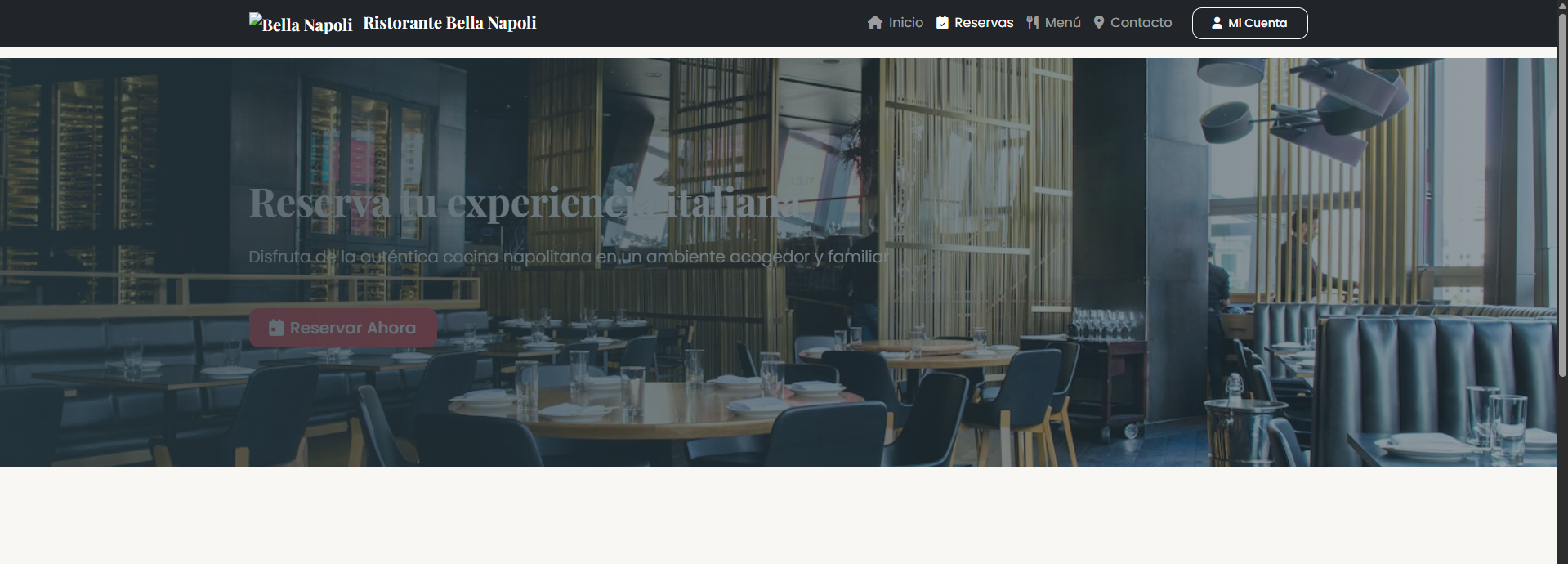


Cuando el usuario accede a esta sección, el sistema le guía para reservar en 4 sencillos pasos. El primer paso, en el que se encuentra, es seleccionar la fecha y el horario de su reserva:

1. Seleccionar una fecha  
    El usuario ve un calendario simplificado. Puede hacer clic en el campo de fecha (por defecto aparece la fecha actual o próxima disponible) para abrir un selector de fechas.  
    El sistema le permite elegir el día en que desea asistir al restaurante.
2. Ver disponibilidad de horarios  
    Una vez seleccionada la fecha, el usuario puede visualizar los horarios disponibles para ese día:  
   * Los horarios activos (en fondo blanco o resaltados) están disponibles para reservar.
   * Los horarios deshabilitados o en gris están no disponibles (ya ocupados o cerrados).
   * Un sistema de colores acompaña la disponibilidad:  
     + Verde: horario disponible.
     + Rojo o gris claro: horario no disponible.
3. Seleccionar un horario  
    El usuario hace clic sobre el horario que desea. El botón seleccionado se resalta (en este caso, en color rojo).
4. Continuar al siguiente paso  
    Luego de seleccionar el horario, simplemente presiona el botón "Siguiente" (rojo con ícono de flecha) ubicado en la esquina inferior derecha para avanzar a Seleccionar Mesa (Paso 2 del proceso).

## 

## Estructuración de Contenido

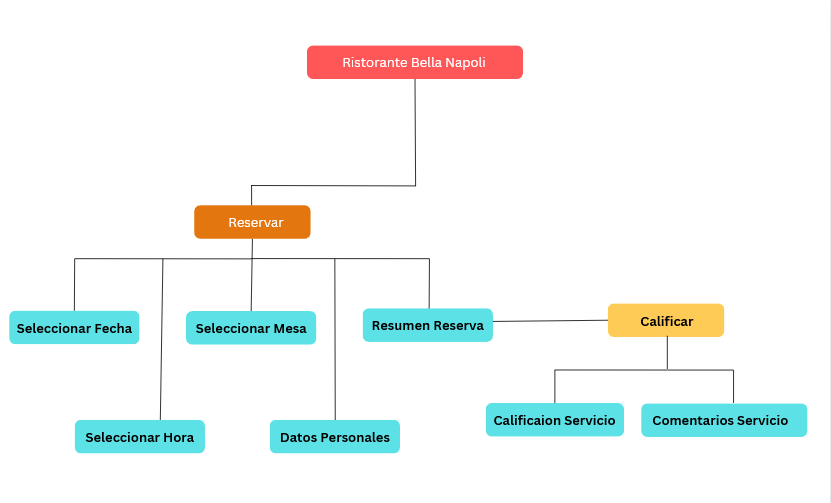


La estructura de navegación de Ristorante Bella Napoli quedó definida así:

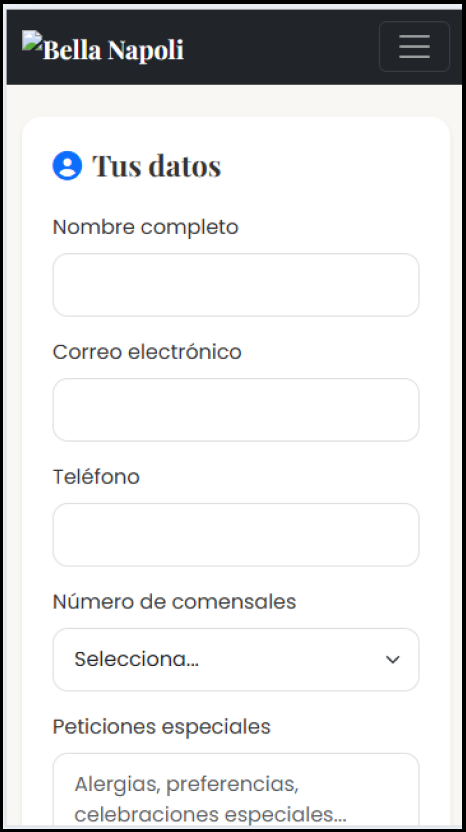
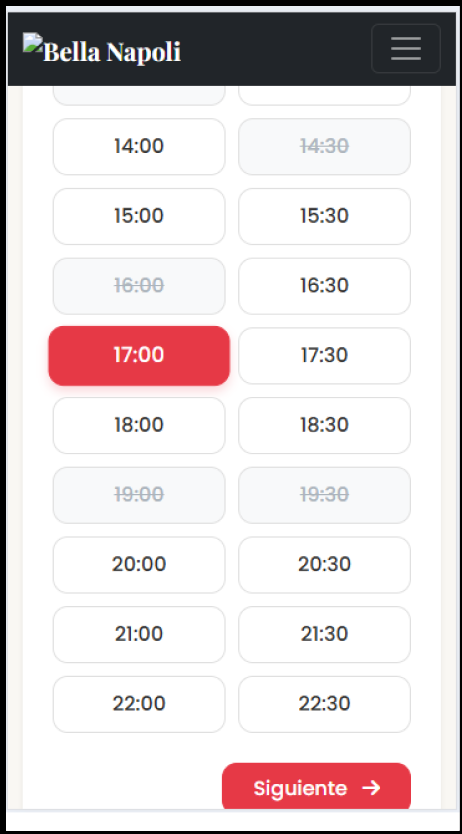
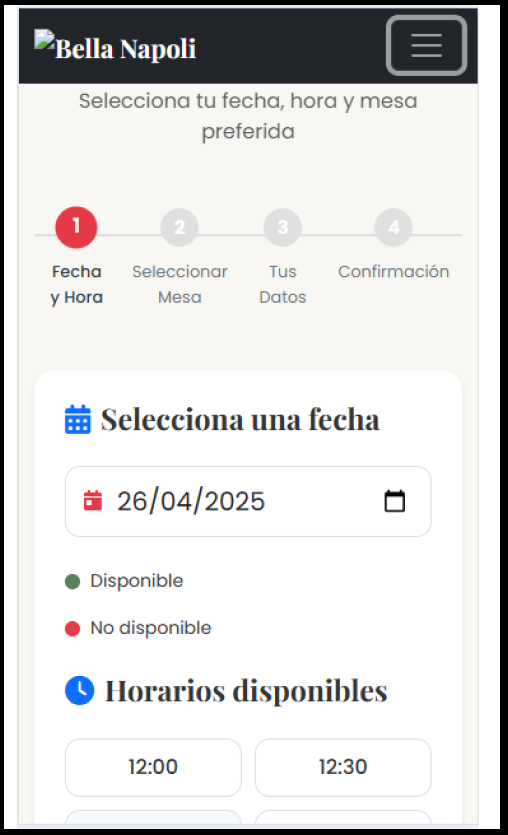
* Inicio
* Reservas  
  + Fecha y Hora
  + Seleccionar Mesa
  + Ingresar Datos
  + Confirmación de Reserva
* Menú
* Contacto
* Mi Cuenta

Esta organización permite a los usuarios acceder rápidamente al proceso de reserva, consultar el menú, contactar con el restaurante o gestionar su cuenta personal, ofreciendo una experiencia de navegación simple, intuitiva y eficiente.

## Site MAP



## Wireflows



El flujo principal de usuario en el sistema de reservas del Proyecto fue cuidadosamente diseñado para ofrecer una experiencia fluida, ordenada y optimizada para dispositivos móviles. El proceso sigue los siguientes pasos:

1. Inicio de Reserva → El usuario comienza seleccionando una fecha disponible desde un calendario interactivo, seguido de la elección de un horario entre los turnos abiertos para ese día, visualizados de forma clara con indicadores de disponibilidad.
2. Selección de Mesa → Una vez elegida la hora, el usuario accede a una pantalla donde puede seleccionar la mesa de su preferencia entre múltiples opciones numeradas, asegurando personalización y disponibilidad en tiempo real.
3. Ingreso de Datos del Usuario → Posteriormente, el usuario completa un breve formulario donde proporciona su nombre completo, correo electrónico, teléfono de contacto y el número de comensales que lo acompañarán. También puede añadir peticiones especiales como alergias, celebraciones o requerimientos particulares.
4. Calificación de la Experiencia → Antes de finalizar, se invita al usuario a calificar el proceso de reserva mediante un sistema de estrellas y, opcionalmente, dejar un comentario sobre su experiencia de uso, lo cual permite retroalimentación continua para la mejora del servicio.
5. Confirmación de Reserva → Finalmente, el usuario visualiza un resumen completo de su reserva (fecha, hora, mesa asignada y número de personas), y recibe un mensaje de confirmación indicando que la reserva fue procesada exitosamente y que los detalles fueron enviados a su correo electrónico.

Este wireflow garantiza un proceso de reserva ágil, intuitivo y orientado al usuario, minimizando tiempos de espera, reduciendo errores de ingreso de datos, y mejorando significativamente la satisfacción del cliente en plataformas móviles.

# CONCLUSIONES

El documento describe el desarrollo e implementación de una plataforma de reservas, destacando el uso de varios patrones de diseño, microservicios y tecnologías de contenedorización. Este proyecto se concibe como una respuesta integral a las necesidades del mercado actual, donde la eficiencia y la accesibilidad en la gestión de reservas son primordiales. El objetivo central es proporcionar una solución unificada que permita a los usuarios gestionar de manera eficiente sus necesidades de reserva de espacios y servicios, abarcando desde parqueaderos y restaurantes hasta la adquisición de boletería y el acceso a bares, todo ello dentro de una misma plataforma cohesionada.

La arquitectura de la plataforma está cuidadosamente diseñada para satisfacer las necesidades tanto de los usuarios como de los proveedores de servicios, reconociendo que el éxito de cualquier sistema de reservas depende de la capacidad de ofrecer valor a ambas partes. Para los usuarios, esto se traduce en una experiencia de reserva intuitiva y optimizada, caracterizada por la facilidad de uso, la rapidez en la navegación y la claridad en la presentación de la información. Se busca eliminar fricciones y simplificar el proceso de reserva, permitiendo a los usuarios encontrar y asegurar los servicios que necesitan con el mínimo esfuerzo posible.

Para los proveedores de servicios, la plataforma ofrece herramientas para gestionar sus inventarios, disponibilidad y reservas de manera eficiente, optimizando la asignación de recursos y maximizando la ocupación. La plataforma facilita la comunicación con los clientes, automatiza tareas administrativas y proporciona información valiosa para la toma de decisiones.

En esencia, el proyecto busca crear un ecosistema de reservas donde la tecnología se convierta en un facilitador, permitiendo a usuarios y proveedores conectarse de manera eficiente y transparente.

# BIBLIOGRAFÍA

Refactoring.Guru: Singleton: <https://refactoring.guru/es/design-patterns/singleton>

Equipo editorial de IONOS. Ionos. Patrón de diseño Factory: las claves del patrón Factory Method: <https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/patron-factory/>

Parra, D. The Power Ups. Patrones de diseño – Observer: <https://thepowerups-learning.com/patrones-de-diseno-observer/>

Amazon. ¿Qué es Docker?: <https://aws.amazon.com/es/docker/>

IBM. ¿Qué es Kubernetes? <https://www.ibm.com/es-es/topics/kubernetes>

GitHub. (n.d). Repositorio: <https://github.com/StevenMadera/Ristorante-Bella-Napoli>

RedHat. (2023, febrero 7). ¿Qué son y para qué sirven los microservicios? <https://www.redhat.com/es/topics/microservices>

Figma. (n.d.). WireFlows para Ristorante-Bella-Napoli: <https://www.figma.com/design/kFNcNlLAdIsE5C3s8JKAeP/WireFlows-para-Ristorante-Bella-Napoli?node-id=0-1&p=f&t=RBppxX0fSLrlcVOp-0>