

[**Etapa 1: Análisis y Planificación 3**](#_heading=h.86w9qdumbsr8)

[● Relevamiento de necesidades (resumen): 3](#_heading=h.4akinsol3lfz)

[● Alcance y requerimientos: 3](#_heading=h.rmqw6u5p27oj)

[● Diseño mediante Casos de Uso y Diagrama de Entidad-Relación: 3](#_heading=h.en2yjemljvaq)

[DER propuesto: 5](#_heading=h.de9r72lxp179)

[● División de tareas: 5](#_heading=h.jl4cuymphyw4)

[**Etapa 2: Propuesta de Implementación Técnica 7**](#_heading=h.t3i79539wihi)

[● Selección e instalación del servidor en Data Center 7](#_heading=h.gsjgx96x5c8t)

[● Software a instalar: 7](#_heading=h.fc6egs95khbc)

[● Motor de Base de Datos: 8](#_heading=h.b7t2uhsyphfu)

[● Tecnologías seleccionadas por capa: 8](#_heading=h.ir4hm12buo7e)

[Desarrollo de prototipo de la APP web del carrito de compras: 9](#_heading=h.imtq2x7kaskx)

[● División de roles: 9](#_heading=h.2p7149mntuos)

[Registro del Desarrollo de la App: 10](#_heading=h.w3w3yvdde498)

[1. Organización Inicial: 10](#_heading=h.r6z9vz411gga)

[2. Desarrollo de Interfaz Inicial: 10](#_heading=h.rzvqlc4l16s)

[3. Creación de la estructura HTML (código final) - /Frontend/index.html: 11](#_heading=h.i0i21rxnutx1)

[4. Estilización del HTML (incluye responsive) - /Frontend/css/style.css: 16](#_heading=h.444c3otxt3xw)

[5. Lógica principal (Frontend) - /Frontend/JS/main.js: 20](#_heading=h.895gg4tuokpe)

[6. Lógica del carrito - /Frontend/JS/carrito.js: 22](#_heading=h.8krhso41e3l8)

[7. Lógica de la interfaz del carrito - /Frontend/JS/carrito-ui.js: 25](#_heading=h.i4xz9r4d493m)

[8. Lógica principal del backend - /Backend/src/index.js: 28](#_heading=h.vkmvk4trmike)

[9. Función para cargar el esquema sql desde backend - /Backend/src/cargarsql.js: 30](#_heading=h.uv3d7kac6i5j)

[10. Endpoint para cargar paquetes desde la base de datos - /Backend/src/routes/paquetes.routes.js: 31](#_heading=h.wbaqv2jwf4xh)

[11. Configuración de la BD - /Backend/src/config/db.js: 33](#_heading=h.tgfsi25n15di)

[12. Configuración de entorno - /Backend/.env: 35](#_heading=h.b5pq5msvl8s4)

[13. Esquema de BD /Backend/src/db/DB.sql: 35](#_heading=h.gsyl8whhc9a4)

[14. Vista con paquetes cargados desde base de datos y la función del carrito(vista del cliente): 41](#_heading=h.l8t2b5x5ald1)

[15. Vista de la página de empleado(jefe de ventas): 42](#_heading=h.qdfoelr68oqy)

[**Manual de Instalación 43**](#_heading=h.nzsbtvtf1w6j)

[1. Requisitos Previos 43](#_heading=h.v700kx5mdx8w)

[2. Clonación o descarga del proyecto 43](#_heading=h.5om8k83hfkvo)

[3. Instalación del Backend 43](#_heading=h.9bn6cpp5lifu)

[4. Ejecución del Frontend 44](#_heading=h.hloxbgxs1y3z)

[**Manual de uso:Cliente 44**](#_heading=h.ew0u8tfylcof)

[● Acceso al sitio 44](#_heading=h.30jtm4qwnoub)

[● Navegación 44](#_heading=h.l5pkdyahdc1u)

[● Carrito de compras 44](#_heading=h.tm6owhd3y8tw)

[**Manual de Uso: Empleado (Jefe de Ventas) 44**](#_heading=h.3jyidm5d73p8)

[● Acceso al sistema 44](#_heading=h.e7n8ppn0ynxc)

[● Menú de gestión de paquetes turísticos 45](#_heading=h.p2t55hotcv69)

[● Visualización de paquetes 45](#_heading=h.oe56a1q3pqgh)

[**Anexo 47**](#_heading=h.8m9jsgcc7j6a)

[**Reflexión : 49**](#_heading=h.ximszjyijs2d)

[**Bibliografía: 50**](#_heading=h.bnrplja9v3qx)

# Etapa 1: Análisis y Planificación

## Relevamiento de necesidades (resumen):

En una entrevista simulada con el cliente (director de la agencia) para desarrollar el carrito de su página web. Este tendrá como propósito captar clientes potenciales interesados en adquirir paquetes turísticos, destacando atributos diferenciales como el enfoque personalizado y propuestas exclusivas.

El carrito ofrecerá paquetes turísticos nacionales e internacionales, con opciones de personalización mediante un sistema modular. Antes de realizar la compra, el usuario accede a información detallada del paquete: itinerarios, fotos, mapas, y políticas de cancelación. Además, se incorporarán reseñas y testimonios que fortalezcan la credibilidad del sitio.

El diseño de navegación incluirá secciones como inicio, destinos, contacto y faq. La experiencia del usuario será optimizada mediante buscadores con filtros, registro y seguimiento de reservas. El diseño será responsive y respetará la identidad visual de la marca (W3Schools, n.d.-a).

En términos de soporte, se considerará la inclusión de chat en línea, formularios de contacto y secciones de ayuda. Para las transacciones, se integrarán métodos de pago como tarjetas de crédito y transferencias bancarias, cumpliendo con estándares de seguridad como el cifrado SSL (SSL Corp., 2023).

También se evaluará la integración con sistemas de gestión de clientes y ventas, y se ofrecerá soporte técnico post-lanzamiento. Se garantizará la protección de datos del usuario, cumpliendo con normativas legales aplicables (Solanki, 2025).

## Alcance y requerimientos:

* + Registro, login, logout y recuperación de contraseña.
  + Edición de perfil básico.
  + Catálogo de paquetes turísticos con filtros simples y buscador.
  + Visualización detallada de paquetes.
  + Agregar, eliminar y modificar elementos del carrito.
  + Formulario de pago validado + generación de orden de compra.
  + Envío de confirmaciones por correo electrónico (cliente y empresa).
  + Historial de compras por usuario.
  + Seguridad básica: SSL, protección de datos.
  + Responsive design (básico).

## Diseño mediante Casos de Uso y Diagrama de Entidad-Relación:

Actores: Usuario (cliente), Sistema (aplicación web) y Administrador

Casos:

* Registro de usuario: El usuario crea una cuenta con correo y contraseña. Se valida la información y se almacena.
* Login / Logout: El usuario inicia sesión para acceder al sistema. Puede cerrar sesión para seguridad.
* Recuperación de contraseña: El usuario solicita la recuperación mediante correo. Se envía enlace seguro para restablecer contraseña.
* Navegar catálogo de paquetes: El usuario visualiza paquetes disponibles. Puede usar filtros y buscador para encontrar paquetes.
* Visualizar detalle de paquete: Mostrar descripción, precio, fechas, servicios incluidos, etc.
* Agregar paquete al carrito: El usuario añade uno o varios paquetes al carrito.
* Modificar carrito: El usuario puede eliminar o cambiar cantidad de paquetes en carrito.
* Realizar pago: El usuario ingresa datos de pago y confirma la compra. Se genera orden y se actualiza el estado.
* Enviar confirmaciones por correo: Se envía mail al usuario con detalles de compra. Se envía mail al sector interno (ventas/administración).
* Ver historial de compras: El usuario puede consultar sus compras anteriores

## DER propuesto:

## 

## 

## División de tareas:

## 

1. **Acosta Lionel** – Desarrollo Frontend: Lionel fue el principal responsable del diseño visual y de la experiencia de usuario del sitio web. Se encargó de:
   1. Diseñar la página principal utilizando HTML, CSS y JavaScript.
   2. Implementar el carrusel de imágenes, las tarjetas dinámicas de paquetes y la barra de navegación.
   3. Garantizar que el sitio fuera responsivo, adaptándose correctamente a distintos tamaños de pantalla.
   4. Aplicar estilos visuales que reflejaran una estética moderna y profesional.

Su aporte fue fundamental para lograr una interfaz amigable, intuitiva y visualmente atractiva para los usuarios.

1. **Favrat Matías** – Líder de Proyecto y Analista Funcional: Matías lideró la planificación general del proyecto, organizando el trabajo del grupo y asegurando que todas las partes técnicas y documentales se completaran correctamente. Sus responsabilidades incluyeron:
   1. Supervisar el desarrollo y la integración entre las distintas capas del sistema.
   2. Trabajar con la base de datos para gestionar el flujo de datos desde y hacia el backend.
   3. Redactar y revisar gran parte de la documentación técnica y del manual de uso.
   4. Coordinar la creación del diagrama entidad-relación y los casos de uso.

Matías fue el nexo entre los aspectos técnicos y organizativos, guiando al grupo en la toma de decisiones clave.

1. **Kettler Thomas** – Backend y Base de Datos: Thomas fue el encargado principal del desarrollo del backend y la base de datos relacional, realizando tareas como:
   1. Crear, configurar y poblar las tablas necesarias en PostgreSQL.
   2. Escribir las consultas SQL, manejar relaciones entre entidades y aplicar validaciones.
   3. Desarrollar la lógica del servidor usando Node.js y Express, incluyendo rutas y controladores.
   4. Asegurar que los datos se almacenaran correctamente y que el sistema fuese seguro y eficiente.
2. Su trabajo fue clave para garantizar la correcta funcionalidad del carrito de compras y la integridad de los datos.

# Etapa 2: Propuesta de Implementación Técnica

## Selección e instalación del servidor en Data Center

| Integrante | Rol Principal |
| --- | --- |
| Favrat Matías | Gestión del proyecto y Analista Funcional |
| Acosta Lionel | Desarrollo Frontend |
| Kettler Thomas | Desarrollo Backend y Base de Datos |

Se optó por Ubuntu Server como sistema operativo principal, dadas sus características de estabilidad, soporte a largo plazo y amplia documentación para entornos de producción (Ubuntu Server Documentation, n.d.). La arquitectura de despliegue contempla dos entornos diferenciados:

* Servidor de desarrollo: destinado a pruebas internas, control de calidad (QA) y preproducción. Aquí se testean funcionalidades antes de pasar al entorno final.
* Servidor de producción: será un entorno seguro, optimizado, aislado del desarrollo y accesible únicamente con de un usuario interno de la compañía (jefe de ventas) con acceso a la aplicación del lado de la empresa con los permisos necesarios para acceder a las funciones de este perfil.. Este entorno será utilizado exclusivamente para el despliegue de la versión final estable del sitio.

Esta división permite un flujo de trabajo ágil, minimizando riesgos durante actualizaciones y cambios.

## Software a instalar:

Como servidor web, se empleará Nginx, conocido por su rendimiento, capacidad de manejo de múltiples conexiones simultáneas y funciones como balanceo de carga y proxy inverso (Bharti, 2024). Para la lógica de aplicación, se utilizará Node.js junto con su gestor de paquetes npm, por su rapidez, asincronía y comunidad activa (Index | Node.js, n.d.; Bukhari, 2024).

En cuanto a la seguridad, se instalarán certificados SSL gratuitos mediante Let 's Encrypt, cumpliendo con los estándares actuales de cifrado de datos (SSL Corp., 2023). Además, se configurarán puertos seguros y se aplicarán políticas de firewall usando herramientas como UFW o iptables, asegurando el control de tráfico entrante y saliente.

## Motor de Base de Datos:

Se seleccionó PostgreSQL como motor de base de datos por su fiabilidad, consistencia y alta capacidad de manejo de grandes volúmenes de información (Instaclustr, 2025). Esta tecnología es especialmente adecuada para proyectos escalables y críticos como el presente.

Se tendrá en cuenta la necesidad de procesar decenas de miles de transacciones diarias sin degradación en el rendimiento. Por ello, se implementarán backups automáticos diarios sobre los principales esquemas: usuarios, paquetes, carrito, pagos y correos electrónicos.

La instalación se basará en PostgreSQL versión 15 o superior. Se creará un rol específico para la aplicación, con privilegios mínimos necesarios, siguiendo el principio de menor privilegio. Para mayor seguridad, el acceso remoto a la base de datos estará restringido únicamente a IPs confiables, reduciendo la exposición a accesos no autorizados.

## Tecnologías seleccionadas por capa:

| Capa | Tecnología | Nota |
| --- | --- | --- |
| Frontend | HTML + CSS + BOOTSTRAP | Responsive y rápido para prototipar |
| Backend | Node.js + Express.js | Rápido y flexible |
| Base de Datos | PostgreSQL | Robusto y eficiente |
| Infraestructura | Nginx + Node.js | Combo muy utilizado en las app web |
| Mail | Nodemailer (Node.js) | Para envíos de mails desde el backend |

## Desarrollo de prototipo de la APP web del carrito de compras:

En su mayoría,el proyecto logró con éxito la integración entre frontend, backend y base de datos,exceptuando situaciones puntuales,el mismo permite la manipulación de datos en tiempo real ofreciendo una experiencia funcional de carrito de compras con paquetes turísticos.La aplicación refleja una buena organización del equipo de trabajo y un enfoque colaborativo durante todas las etapas del desarrollo.

Dado el tiempo disponible y los objetivos planteados para esta instancia, se priorizó la implementación de un prototipo funcional que abarca las funcionalidades principales solicitadas por la consigna.

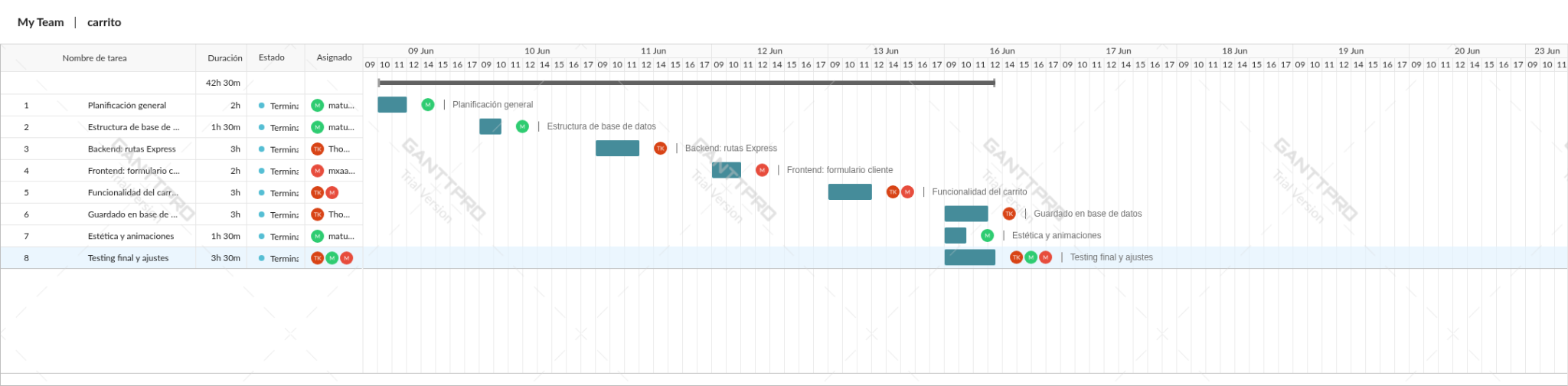
Algunas funcionalidades solicitadas en la consigna (como la función de editar paquetes desde la vista de empleado) no pudieron concretarse en esta etapa, aunque sí fueron consideradas en el diseño general de la arquitectura.

El prototipo presentado cumple con los objetivos mínimos del proyecto y sienta las bases para una posible evolución hacia una aplicación completa, incorporando las funciones pendientes y optimizando su rendimiento y seguridad.

## División de roles:

| Integrante | Rol Principal | Colaboración Adicional |
| --- | --- | --- |
| Favrat Matías | Gestión del proyecto y Analista Funcional | Apoyo en diseño de base de datos, frontend y documentación técnica. |
| Acosta Lionel | Desarrollo Frontend | Apoyo en testing y documentación de usuario. |
| Kettler Thomas | Desarrollo Backend y Base de Datos | Apoyo en documentación y gestión . |

## Registro del Desarrollo de la App:



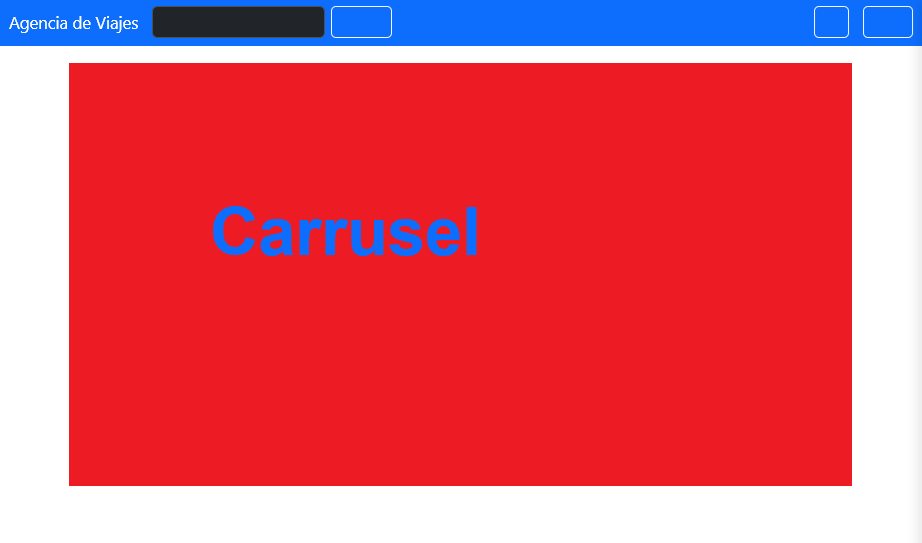
[¡ Aquí está el diagrama de Gantt !](https://app.ganttpro.com/shared/token/f98c89730a869957f0c4cc5e302f184f496764ea9a5b7235c2b91f9ed3c57ca0/1825826)

### Organización Inicial:

Una vez repartidas las tareas se creó una carpeta compartida en Google Drive para respaldar y versionar archivos HTML, CSS, JavaScript, documentación y otros recursos. Luego se estructuró la carpeta dividiéndola en Frontend y Backend con sus respectivos archivos.

### Desarrollo de Interfaz Inicial:

En el software de diseño Figma se boceto la vista de la página principal donde se podrá apreciar un carrusel con fotos de lugares turísticos, debajo se observan los paquetes disponibles obtenidos desde la base de datos, en la barra de navegación se nota una barra de búsqueda junto a su botón y por último, a la derecha se ubican los botones para abrir el carrito y para visualizar los datos del usuario.



### Creación de la estructura HTML (código final) - /Frontend/index.html:

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Agencia de Viajes - Página Principal</title>

<!-- Hojas de estilo y recursos externos →

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/css/bootstrap.min.css" rel="stylesheet" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/bootstrap-icons.css" />

<!-- Hoja de estilo personalizada -->

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<header>

<nav class="navbar navbar-expand-lg bg-primary" data-bs-theme="dark">

<div class="container-fluid">

<a class="navbar-brand" href="./index.html">Agencia de Viajes</a>

<!-- Botón para menú colapsable en móviles -->

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarContent"><span

class="navbar-toggler-icon"></span></button>

<div class="collapse navbar-collapse" id="navbarContent">

<!-- Barra de búsqueda -->

<form class="d-flex me-auto" role="search">

<input class="form-control me-2" type="search" placeholder="Buscar paquetes..." />

<button class="btn btn-outline-light" type="submit">

Buscar

</button>

</form>

<!-- Icono del carrito con contador -->

<button class="btn btn-outline-light position-relative me-3" id="btn-carrito-header">

<i class="bi bi-cart-fill"></i>

<span

class="position-absolute top-0 start-100 translate-middle badge rounded-pill bg-danger pop-notification"

id="badge-carrito">

0

</span>

</button>

<!-- Menú desplegable de usuario -->

<div class="dropdown">

<button class="btn btn-outline-light dropdown-toggle" type="button" id="userDropdown"

data-bs-toggle="dropdown">

<i class="bi bi-person-circle"></i>

</button>

<ul class="dropdown-menu dropdown-menu-end">

<li>

<a class="dropdown-item" href="cuenta\_usuario.html">Mi Cuenta</a>

</li>

<li>

<hr class="dropdown-divider" />

</li>

<li>

<button class="dropdown-item" id="logoutBtn">

Cerrar Sesión

</button>

</li>

</ul>

</div>

</div>

</div>

</nav>

</header>

<!-- Carrusel -->

<section class="container mt-4">

<div id="carouselPrincipal" class="carousel slide mb-4" data-bs-ride="carousel">

<div class="carousel-inner">

<!-- Imágenes del carrusel -->

<div class="carousel-item active">

<img src="img/peru.jpg" alt="Imagen 1" class="img-carousel" />

</div>

<div class="carousel-item">

<img src="img/carousel\_img2.jpg" alt="Imagen 2" class="img-carousel" />

</div>

<div class="carousel-item">

<img src="img/espana.webp" alt="Imagen 3" class="img-carousel" />

</div>

<div class="carousel-item">

<img src="img/puerto\_rico.webp" class="img-carousel" alt="Imagen 4" />

</div>

</div>

<!-- Controles del carrusel -->

<button class="carousel-control-prev" type="button" data-bs-target="#carouselPrincipal" data-bs-slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="visually-hidden">Anterior</span>

</button>

<button class="carousel-control-next" type="button" data-bs-target="#carouselPrincipal" data-bs-slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="visually-hidden">Siguiente</span>

</button>

</div>

</section>

<section id="galeria-productos" class="container mt-4">

<!-- Las tarjetas de productos se generarán dinámicamente aquí -->

</section>

<!-- Panel del carrito -->

<aside id="panel-carrito" class="carrito-panel">

<!-- Encabezado del carrito -->

<div class="carrito-header bg-light">

<h4 class="mb-0">

<i class="bi bi-cart3 me-2"></i>Tu Carrito

<span class="badge bg-primary ms-2" id="contador-carrito">0</span>

</h4>

<button id="cerrar-carrito" class="btn btn-sm btn-outline-secondary">

<i class="bi bi-x-lg"></i>

</button>

</div>

<!-- Cuerpo del carrito -->

<div class="carrito-body">

<div id="lista-carrito" class="list-group list-group-flush">

<!-- Items del carrito se agregarán dinámicamente aquí -->

</div>

<!-- Mensaje cuando el carrito está vacío -->

<div class="carrito-vacio text-center py-4" id="carrito-vacio">

<i class="bi bi-cart-x text-muted" style="font-size: 2rem"></i>

<p class="mt-2 text-muted">Tu carrito está vacío</p>

</div>

</div>

<!-- Pie del carrito con total y botón de pago -->

<div class="carrito-footer">

<div class="d-flex justify-content-between align-items-center mb-3">

<h5 class="mb-0">Total:</h5>

<h4 class="mb-0 text-primary" id="total-carrito">$0.00</h4>

</div>

<button id="proceder-pago" class="btn btn-primary w-100 py-2" disabled>

<i class="bi bi-credit-card me-2"></i>Proceder al pago

</button>

</div>

</aside>

<!-- Script de autenticación -->

<script type="module">

import { logout } from "./js/auth.js";

const userName = userData.client\_name || 'Usuario';

document.addEventListener("DOMContentLoaded", () => {

const userDropdown = document.getElementById("userDropdown");

const logoutBtn = document.getElementById("logoutBtn");

const userData = JSON.parse(localStorage.getItem("userData"));

if (userData) {

userDropdown.innerHTML = `<i class="bi bi-person-circle"></i> ${userData.name}`;

}

logoutBtn.addEventListener("click", logout);

});

</script>

<!-- Bootstrap JS -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/js/bootstrap.bundle.min.js"></script>

<!-- Script principal de la aplicación -->

<script type="module" src="js/main.js"></script>

<!-- Script para manejo de usuario -->

<script>

document.addEventListener("DOMContentLoaded", function () {

// Verificación de autenticación

document.addEventListener("DOMContentLoaded", () => {

const token = localStorage.getItem('authToken');

if (!token) {

document.getElementById('userDropdown').style.display = 'none';

}

});

// Manejo de datos del usuario

const userData = JSON.parse(localStorage.getItem('userData'));

const userDropdown = document.getElementById('userDropdown');

if (userData && userDropdown) {

// Configuración del dropdown de usuario con datos del localStorage

const userName = userData.client\_name || userData.name || 'Usuario';

const userEmail = userData.client\_email || userData.email;

const userIcon = userDropdown.querySelector('i.bi-person-circle') ||

document.createElement('i');

userIcon.className = 'bi bi-person-circle';

userDropdown.innerHTML = '';

userDropdown.appendChild(userIcon);

// Agregar imagen de perfil si existe

if (userData.profileImg) {

const profileImg = document.createElement('img');

profileImg.src = userData.profileImg;

profileImg.className = 'profile-img';

profileImg.alt = userName;

userDropdown.insertBefore(profileImg, userIcon);

}

// Agregar nombre de usuario (visible en desktop)

const nameSpan = document.createElement('span');

nameSpan.className = 'ms-2 d-none d-md-inline';

nameSpan.textContent = userName;

userDropdown.appendChild(nameSpan);

}

// Manejo de logout

const logoutBtn = document.getElementById('logoutBtn');

if (logoutBtn) {

logoutBtn.addEventListener('click', function (e) {

e.preventDefault();

localStorage.removeItem('userData');

window.location.href = 'index.html';

});

}

});

</script>

</body>

</html>

### Estilización del HTML (incluye responsive) - /Frontend/css/style.css:

.carrito-panel {

position: fixed;

top: 0;

right: 0;

width: 380px;

/\* Ancho fijo para desktop \*/

height: 100vh;

/\* Ocupa toda la altura de la pantalla \*/

background: white;

box-shadow: -5px 0 15px rgba(0, 0, 0, 0.1);

z-index: 1050;

/\* Asegura que esté por encima de otros elementos \*/

transform: translateX(100%);

/\* Inicialmente oculto \*/

transition: transform 0.3s cubic-bezier(0.4, 0, 0.2, 1);

/\* Animación suave \*/

display: flex;

flex-direction: column;

border-left: 1px solid #e0e0e0;

}

.carrito-panel.abierto {

transform: translateX(0);

/\* Muestra el panel \*/

}

.carrito-header {

padding: 1.25rem;

border-bottom: 1px solid #e0e0e0;

display: flex;

justify-content: space-between;

align-items: center;

background: #f5f5f5;

/\* Fondo gris claro \*/

}

.carrito-body {

flex: 1;

/\* Ocupa todo el espacio disponible \*/

overflow-y: auto;

/\* Scroll vertical si es necesario \*/

padding: 0;

}

.carrito-footer {

border-top: 1px solid #e0e0e0;

padding: 1.25rem;

background: #f5f5f5;

}

.item-carrito {

display: flex;

align-items: center;

padding: 1rem;

border-bottom: 1px solid #eee;

transition: background-color 0.2s;

/\* Efecto hover suave \*/

}

.item-carrito:hover {

background-color: #f9f9f9;

/\* Cambio de color al pasar el mouse \*/

}

.item-carrito img {

width: 60px;

height: 60px;

object-fit: cover;

/\* Asegura que la imagen cubra el espacio \*/

border-radius: 4px;

margin-right: 1rem;

border: 1px solid #e0e0e0;

}

.item-info {

flex: 1;

/\* Ocupa todo el espacio restante \*/

min-width: 0;

/\* Evita desbordamiento \*/

}

.item-info h6 {

margin-bottom: 0.25rem;

white-space: nowrap;

/\* Evita saltos de línea \*/

overflow: hidden;

text-overflow: ellipsis;

/\* Puntos suspensivos si el texto es muy largo \*/

}

.item-info small {

color: #666;

/\* Texto secundario más claro \*/

}

.item-controls {

display: flex;

align-items: center;

}

.cantidad-input {

width: 50px;

text-align: center;

margin: 0 0.5rem;

padding: 0.25rem;

}

.img-carousel {

height: 500px;

width: 100%;

object-fit: cover;

/\* Asegura que las imágenes cubren el espacio \*/

}

.profile-img {

width: 32px;

height: 32px;

border-radius: 50%;

/\* Forma circular \*/

object-fit: cover;

border: 2px solid rgba(255, 255, 255, 0.8);

margin-right: 8px;

}

.profile-section {

background-color: #f8f9fa;

border-radius: 10px;

padding: 2rem;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

#profile-picture {

object-fit: cover;

border: 3px solid #0d6efd;

/\* Borde azul \*/

border-radius: 50%;

}

.pop-notification {

animation: pulse 0.5s ease-in-out;

/\* Animación de latido \*/

}

@keyframes pulse {

0% {

transform: scale(1);

}

50% {

transform: scale(1.2);

}

/\* Efecto de agrandamiento \*/

100% {

transform: scale(1);

}

}

/\* Diseño responsive \*/

@media (max-width: 768px) {

.carrito-panel {

width: 100%;

/\* Ocupa todo el ancho en móviles \*/

border-left: none;

}

body.carrito-abierto {

overflow: hidden;

/\* Evita scroll cuando el carrito está abierto \*/

}

.item-carrito {

padding: 0.75rem;

/\* Menos padding en móviles \*/

}

.item-carrito img {

width: 50px;

height: 50px;

/\* Imágenes más pequeñas en móviles \*/

}

}

### Lógica principal (Frontend) - /Frontend/JS/main.js:

import { carrito } from './carrito.js';

import { CarritoUI } from './carrito-ui.js';

document.addEventListener('DOMContentLoaded', () => {

new CarritoUI(); // Inicializa la interfaz del carrito

cargarProductos(); // Carga los productos al iniciar

});

// Carga los productos desde la API y los renderiza en la página. Muestra un mensaje de error si falla la carga.

async function cargarProductos() {

try {

const response = await fetch('http://localhost:3000/api/paquetes');

if (!response.ok) {

throw new Error(`Error HTTP: ${response.status}`);

}

const productos = await response.json();

renderizarProductos(productos);

} catch (error) {

console.error('Error al cargar productos:', error);

mostrarErrorAlUsuario('No se pudieron cargar los paquetes');

}

}

// Renderiza una lista de productos en el contenedor especificado. @param {Array} productos - Lista de productos a renderizar.

function renderizarProductos(productos) {

const contenedor = document.getElementById('galeria-productos');

if (!contenedor) return; // Asegurarse de que el elemento existe

contenedor.innerHTML = '';

// Crear una fila para contener todas las tarjetas

const row = document.createElement('div');

row.className = 'row g-3';

productos.forEach(producto => {

const precio = Number(producto.product\_unit\_price) || 0;

// Crear columna para cada producto

const col = document.createElement('div');

col.className = 'col-md-3 mb-4';

// Crea la tarjeta del producto con sus datos e imagen

const card = document.createElement('div');

card.className = 'card h-100';

card.dataset.id = producto.id\_product;

card.dataset.nombre = producto.product\_name;

card.dataset.precio = precio;

card.dataset.imagen = `../img/${producto.product\_image}`;

card.innerHTML = `

<img src="../img/${producto.product\_image}"

class="card-img-top"

alt="${producto.product\_name}">

<div class="card-body">

<h5 class="card-title">${producto.product\_name}</h5>

<p class="card-text">${producto.product\_description}</p>

<p class="text-primary fw-bold">$${precio.toFixed(2)}</p>

<button class="btn btn-primary btn-agregar">Agregar al carrito</button>

</div>

`;

col.appendChild(card);

row.appendChild(col);

});

contenedor.appendChild(row);

}

//Muestra un mensaje de error al usuario en formato toast. param {string} mensaje - Mensaje de error a mostrar.

function mostrarErrorAlUsuario(mensaje) {

const toast = document.createElement('div');

toast.className = 'position-fixed bottom-0 end-0 p-3';

toast.innerHTML = `

<div class="toast show" role="alert">

<div class="toast-header bg-danger text-white">

<strong class="me-auto">Error</strong>

<button type="button" class="btn-close" data-bs-dismiss="toast"></button>

</div>

<div class="toast-body">${mensaje}</div>

</div>

`;

document.body.appendChild(toast);

setTimeout(() => toast.remove(), 5000);

}

### Lógica del carrito - /Frontend/JS/carrito.js:

import { authHeader } from './auth.js';

// Clase que maneja el carrito de compras, incluyendo: Almacenamiento local, Sincronización con el backend, Notificación de cambios

class Carrito {

constructor() {

this.items = JSON.parse(localStorage.getItem('carrito')) || [];

this.subscribers = [];

}

//Registra una función para ser notificada cuando cambia el carrito

subscribe(callback) {

this.subscribers.push(callback);

}

//Notifica a todos los suscriptores y guarda el estado actual en localStorage

notify() {

this.subscribers.forEach(cb => cb(this.items));

localStorage.setItem('carrito', JSON.stringify(this.items));

}

//Agrega un item al carrito o incrementa su cantidad si ya existe

agregarItem(id, nombre, precio, imagen) {

const itemExistente = this.items.find(item => item.id === id);

if (itemExistente) {

itemExistente.cantidad += 1;

} else {

this.items.push({

id,

nombre,

precio: Number(precio) || 0, // Asegurar que sea número

imagen,

cantidad: 1

});

}

this.notify();

}

//Actualiza la cantidad de un item existente

actualizarCantidad(id, nuevaCantidad) {

const item = this.items.find(item => item.id === id);

if (!item) return;

item.cantidad = nuevaCantidad;

this.notify();

}

//Elimina un item del carrito

eliminarItem(id) {

this.items = this.items.filter(item => item.id !== id);

this.notify();

}

//Vacía completamente el carrito

vaciarCarrito() {

this.items = [];

this.notify();

}

//Calcula el total del carrito

get total() {

return this.items.reduce((sum, item) => sum + (item.precio \* item.cantidad), 0);

}

//Calcula la cantidad total de items en el carrito

get cantidadItems() {

return this.items.reduce((sum, item) => sum + item.cantidad, 0);

}

//Sincroniza el carrito local con el backend

async sincronizarBackend() {

try {

const response = await fetch('http://localhost:3000/api/carrito/sincronizar', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

...authHeader()

},

body: JSON.stringify({

items: this.items

})

});

if (!response.ok) {

throw new Error('Error al sincronizar carrito');

}

} catch (error) {

console.error('Error al sincronizar carrito:', error);

throw error;

}

}

//Carga el carrito desde el backend

async cargarDesdeBackend() {

try {

const response = await fetch('http://localhost:3000/api/carrito', {

headers: authHeader()

});

if (!response.ok) {

throw new Error('Error al cargar carrito');

}

const data = await response.json();

this.items = data.carrito || [];

this.notify();

} catch (error) {

console.error('Error al cargar carrito:', error);

throw error;

}

}

}

export const carrito = new Carrito();

// Función para actualizar el contador del carrito en la interfaz

function actualizarContadorCarrito(cantidad) {

const badge = document.getElementById('badge-carrito');

const prevCount = parseInt(badge.textContent) || 0;

badge.textContent = cantidad;

// Only animate when adding items

badge.classList.remove('pop-notification');

void badge.offsetWidth; // Forzar reflow

badge.classList.add('pop-notification');

}

// Suscribir esta función al carrito

carrito.subscribe(items => {

const cantidad = carrito.cantidadItems;

actualizarContadorCarrito(cantidad);

});

### Lógica de la interfaz del carrito - /Frontend/JS/carrito-ui.js:

import { carrito } from './carrito.js';

export class CarritoUI {

constructor() {

// Selección de elementos DOM relevantes

this.panel = document.getElementById('panel-carrito');

this.lista = document.getElementById('lista-carrito');

this.botonCarrito = document.getElementById('btn-carrito-header');

this.botonCerrar = document.getElementById('cerrar-carrito');

this.contador = document.getElementById('contador-carrito');

this.totalElement = document.getElementById('total-carrito');

this.botonPago = document.getElementById('proceder-pago');

this.mensajeVacio = document.getElementById('carrito-vacio');

this.badgeCarrito = document.getElementById('badge-carrito');

this.iniciarEventos();

carrito.subscribe(this.actualizarVista.bind(this));

this.actualizarVista(carrito.items); // Actualizar al iniciar

}

//Configura los event listeners para interacciones del usuario

iniciarEventos() {

// Abrir/cerrar el panel del carrito

this.botonCarrito.addEventListener('click', () => this.abrirCarrito());

this.botonCerrar.addEventListener('click', () => this.cerrarCarrito());

//Procesar pago

this.botonPago.addEventListener('click', () => this.procederAlPago());

// Evento delegado para agregar productos desde la galería

document.addEventListener('click', (e) => {

if (e.target.classList.contains('btn-agregar')) {

const card = e.target.closest('.card');

const id = card.dataset.id;

const nombre = card.dataset.nombre;

const precio = parseFloat(card.dataset.precio);

const imagen = card.querySelector('img').src;

carrito.agregarItem(id, nombre, precio, imagen);

}

});

//Eliminar items y actualizar cantidades

this.lista.addEventListener('click', (e) => {

if (e.target.closest('.btn-eliminar')) {

const id = e.target.closest('.btn-eliminar').dataset.id;

carrito.eliminarItem(id);

}

});

this.lista.addEventListener('change', (e) => {

if (e.target.classList.contains('cantidad-input')) {

const id = e.target.dataset.id;

const nuevaCantidad = parseInt(e.target.value);

if (nuevaCantidad > 0) {

carrito.actualizarCantidad(id, nuevaCantidad);

} else {

e.target.value = 1;

}

}

});

}

abrirCarrito() {

this.panel.classList.add('abierto');

document.body.classList.add('carrito-abierto');

}

cerrarCarrito() {

this.panel.classList.remove('abierto');

document.body.classList.remove('carrito-abierto');

}

procederAlPago() {

if (carrito.items.length === 0) return;

// Simulamos un proceso de pago

setTimeout(() => {

alert(`¡Pago exitoso! Total: $${carrito.total.toFixed(2)}`);

carrito.vaciarCarrito();

this.cerrarCarrito();

}, 1000);

}

actualizarVista(items) {

const listaCarrito = document.getElementById('lista-carrito');

const totalCarrito = document.getElementById('total-carrito');

const carritoVacio = document.getElementById('carrito-vacio');

const contadorCarrito = document.getElementById('contador-carrito');

const botonPago = document.getElementById('proceder-pago');

const badgeCarrito = document.getElementById('badge-carrito');

// Calcular cantidad total de items

const cantidadTotal = items.reduce((sum, item) => sum + item.cantidad, 0);

// Actualizar ambos contadores

contadorCarrito.textContent = cantidadTotal;

badgeCarrito.textContent = cantidadTotal;

if (items.length === 0) {

listaCarrito.innerHTML = '';

carritoVacio.style.display = 'block';

totalCarrito.textContent = '$0.00';

botonPago.disabled = true;

return;

}

carritoVacio.style.display = 'none';

listaCarrito.innerHTML = '';

botonPago.disabled = false;

items.forEach(item => {

const precio = Number(item.precio) || 0;

const subtotal = precio \* item.cantidad;

const li = document.createElement('li');

li.className = 'list-group-item';

li.innerHTML = `

<div class="d-flex justify-content-between align-items-center">

<div class="d-flex align-items-center">

<img src="${item.imagen}" width="50" height="50" class="me-3 rounded">

<div>

<h6 class="mb-0">${item.nombre}</h6>

<small>$${precio.toFixed(2)} c/u</small>

</div>

</div>

<div class="d-flex align-items-center">

<input type="number" min="1" value="${item.cantidad}"

class="form-control form-control-sm cantidad-input me-2"

style="width: 60px" data-id="${item.id}">

<span class="me-3">$${subtotal.toFixed(2)}</span>

<button class="btn btn-sm btn-outline-danger btn-eliminar" data-id="${item.id}">

<i class="bi bi-trash"></i>

</button>

</div>

</div>

`;

listaCarrito.appendChild(li);

});

const total = items.reduce((sum, item) => {

const precio = Number(item.precio) || 0;

return sum + (precio \* item.cantidad);

}, 0);

totalCarrito.textContent = `$${total.toFixed(2)}`;

}

}

### Lógica principal del backend - /Backend/src/index.js:

import express from 'express';

import path from 'path';

import { fileURLToPath } from 'url';

import dotenv from 'dotenv';

import cors from 'cors';

import { securityHeaders, authLimiter } from './securityMiddleware.js';

import { errorHandler } from './middlewares/errorHandler.js';

import carritoRoutes from './routes/carrito.routes.js';

import paquetesRoutes from './routes/paquetes.routes.js';

import usersRoutes from './routes/users.routes.js';

import authRoutes from './routes/auth.routes.js';

import cookieParser from 'cookie-parser';

// Cargar variables de entorno

dotenv.config({ path: `.env.${process.env.NODE\_ENV || 'development'}` });

// Constantes - MOVER ESTO ARRIBA

const PORT = process.env.PORT || 3000;

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

const frontendPath = path.join(\_\_dirname, '../../Frontend');

// Iniciar app

const app = express();

// Middleware

app.use(securityHeaders);

app.use(cors({

origin: [`http://localhost:${PORT}`, 'http://localhost:5500'],

methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],

allowedHeaders: ['Content-Type', 'Authorization'],

credentials: true

}));

// Body parsers

app.use(express.json());

app.use((req, res, next) => {

next();

});

app.use(express.static(frontendPath));

app.use(express.static(path.join(\_\_dirname, '../Frontend')));

app.use((err, req, res, next) => {

if (err.status === 404) {

console.warn(`Static file not found: ${req.url}`);

}

next(err);

});

app.use(cookieParser());

// Rutas

app.use("/api/carrito", carritoRoutes);

app.use("/api/paquetes", paquetesRoutes);

app.use("/api/users", usersRoutes);

app.use("/api/auth", authRoutes);

app.use('/api/auth/login', authLimiter);

// Manejo de errores

app.use(errorHandler);

// Configuración del servidor

app.listen(PORT, () => {

console.log(`🚀 Servidor ejecutándose en el Puerto:${PORT}`);

});

// Manejo de errores no capturados

process.on('unhandledRejection', err => {

console.error('Unhandled Rejection:', err);

process.exit(1);

});

### Función para cargar el esquema sql desde backend - /Backend/src/cargarsql.js:

import fs from 'fs';

import { pool } from './config/db.js';

import bcrypt from 'bcrypt';

async function cargarsql() {

const client = await pool.connect();

try {

await client.query('BEGIN');

// Ejecutar script SQL

const sql = fs.readFileSync('./db/DB.sql', 'utf8');

console.log('📄 Ejecutando script SQL...');

await client.query(sql);

// Hash contraseñas existentes

console.log('🔒 Hasheando contraseñas existentes...');

await hashExistingPasswords(client);

console.log(`🔒 Contraseña del cliente ${client.id\_client} hasheada`);

await client.query('COMMIT');

console.log('✅ Base de datos inicializada correctamente');

} catch (error) {

await client.query('ROLLBACK');

console.error('❌ Error al inicializar la base de datos:', error.message);

throw error; // Fallo al inicializar la base de datos

} finally {

client.release();

await pool.end();

}

}

async function hashExistingPasswords(client) {

// Hash contraseñas de clientes

const clients = await client.query('SELECT id\_client, client\_password FROM client');

for (const client of clients.rows) {

const hashed = await bcrypt.hash(client.client\_password, 10);

await client.query(

'UPDATE client SET client\_password = $1 WHERE id\_client = $2',

[hashed, client.id\_client]

);

}

// Verificar si la contraseña ya está hasheada

if (!client.client\_password.startsWith('$2b$')) {

const hashed = await bcrypt.hash(client.client\_password, 10);

}

// Hash contraseñas de empleados

const employees = await client.query(

'SELECT id\_internal\_user, internal\_user\_password FROM internal\_user'

);

for (const emp of employees.rows) {

const hashed = await bcrypt.hash(emp.internal\_user\_password, 10);

await client.query(

'UPDATE internal\_user SET internal\_user\_password = $1 WHERE id\_internal\_user = $2',

[hashed, emp.id\_internal\_user]

);

}

}

cargarsql()

### Endpoint para cargar paquetes desde la base de datos - /Backend/src/routes/paquetes.routes.js:

import { Router } from 'express';

import { pool } from '../config/db.js';

import { authenticateToken } from '../middlewares/authMiddleware.js';

const router = Router();

// Obtener todos los paquetes

router.get('/', async (req, res) => {

try {

const result = await pool.query('SELECT \* FROM PRODUCT');

res.json(result.rows);

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Error al obtener productos' });

}

});

// Crear nuevo paquete

router.post('/', authenticateToken, async (req, res) => {

const { product\_name, product\_description, product\_unit\_price, product\_image } = req.body;

if (!product\_name || !product\_description || !product\_unit\_price) {

return res.status(400).json({ error: "Faltan campos obligatorios" });

}

//lo agrego matias//

const exists = await pool.query(

"SELECT \* FROM product WHERE product\_name = $1",

[product\_name]

);

if (exists.rows.length > 0) {

return res.status(400).json({ error: "Nombre ya existe" });

}

try {

const result = await pool.query(

`INSERT INTO product

(product\_name, product\_description, product\_unit\_price, product\_image)

VALUES ($1, $2, $3, $4)

RETURNING \*`,

[product\_name, product\_description, product\_unit\_price, product\_image]

);

res.status(201).json(result.rows[0]);

} catch (error) {

console.error("Error al crear paquete:", error);

res.status(500).json({ error: "Error interno del servidor" });

}

});

// Actualizar paquete

router.put('/:id', authenticateToken, async (req, res) => {

const { id } = req.params;

const { product\_name, product\_description, product\_unit\_price, product\_image } = req.body;

try {

const result = await pool.query(

`UPDATE product SET

product\_name = $1,

product\_description = $2,

product\_unit\_price = $3,

product\_image = $4

WHERE id\_product = $5

RETURNING \*`,

[product\_name, product\_description, product\_unit\_price, product\_image, id]

);

if (result.rows.length === 0) {

return res.status(404).json({ error: "Paquete no encontrado" });

}

res.json(result.rows[0]);

} catch (error) {

console.error("Error al actualizar paquete:", error);

res.status(500).json({ error: "Error interno del servidor" });

}

});

// Borrar paquete

router.delete('/:id', authenticateToken, async (req, res) => {

const { id } = req.params;

try {

await pool.query("DELETE FROM product WHERE id\_product = $1", [id]);

res.json({ mensaje: "Paquete eliminado correctamente" });

} catch (error) {

console.error("Error al eliminar paquete:", error);

res.status(500).json({ error: "Error interno del servidor" });

}

});

export default router;

### Configuración de la BD - /Backend/src/config/db.js:

import pg from 'pg';

import dotenv from 'dotenv';

dotenv.config();

const { Pool } = pg;

export const pool = new Pool({

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

host: process.env.DB\_HOST,

port: process.env.DB\_PORT,

database: process.env.DB\_NAME,

max: 20,

idleTimeoutMillis: 30000,

connectionTimeoutMillis: 2000

});

// Opcional: Prueba la conexión al iniciar

pool.query('SELECT NOW()')

.then(() => console.log('✅ PostgreSQL conectado'))

.catch(err => console.error('❌ Error de conexión a PostgreSQL:', err));

async function main() {

try {

const timeResult = await pool.query("SELECT NOW()");

console.log("🕒 Hora actual desde PostgreSQL:", timeResult.rows[0]);

const tablesResult = await pool.query(

`SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public'`);

console.log("🗂️ Tablas:", tablesResult.rows);

} catch (err) {

console.error("❌ Error en la consulta:", err);

} finally {

}

}

if (process.env.RUN\_MIGRATIONS === 'true') {

import('./migrate-passwords.js').then(m => m.default());

}

if (process.env.RUN\_MIGRATIONS === 'true') {

import('./migrate-passwords.js').then(m => m.default());

}

### Configuración de entorno - /Backend/.env:

DB\_USER=postgres

DB\_HOST=localhost

DB\_PASSWORD="12345678"

DB\_NAME=carrito

DB\_PORT=5432

JWT\_SECRET="123456"

PORT=3000

NODE\_ENV=development

JWT\_REFRESH\_SECRET="654321"

### Esquema de BD /Backend/src/db/DB.sql:

-- Database: carrito

-- DROP DATABASE IF EXISTS carrito;

CREATE TABLE CLIENT (

ID\_CLIENT SERIAL PRIMARY KEY,

CLIENT\_NAME VARCHAR(30) NOT NULL,

CLIENT\_LAST\_NAME VARCHAR(30) NOT NULL,

CLIENT\_EMAIL VARCHAR(100) UNIQUE NOT NULL,

CLIENT\_ADDRESS VARCHAR(100) NOT NULL,

CLIENT\_PASSWORD VARCHAR(100) NOT NULL

);

CREATE TABLE PRODUCT (

ID\_PRODUCT SERIAL PRIMARY KEY,

PRODUCT\_NAME VARCHAR(100) NOT NULL,

PRODUCT\_DESCRIPTION VARCHAR(200) NOT NULL,

PRODUCT\_UNIT\_PRICE NUMERIC(12, 2) NOT NULL,

PRODUCT\_IMAGE VARCHAR(200) NOT NULL

);

CREATE TABLE WISH (

ID\_WISH SERIAL PRIMARY KEY,

FK\_CLIENT INT NOT NULL,

WISH\_DATE DATE NOT NULL,

WISH\_STATE VARCHAR(10) NOT NULL,

WISH\_TOTAL NUMERIC(12, 2) NOT NULL,

CONSTRAINT FK\_CLIENT\_WISH FOREIGN KEY (FK\_CLIENT) REFERENCES CLIENT (ID\_CLIENT)

);

CREATE TABLE WISH\_DETAIL (

ID\_WISH\_DETAIL SERIAL PRIMARY KEY,

FK\_WISH INT NOT NULL,

FK\_PRODUCT INT NOT NULL,

WISH\_DETAIL\_AMOUNT INT NOT NULL,

WISH\_DETAIL\_SUBTOTAL NUMERIC(12, 2) NOT NULL,

CONSTRAINT FK\_WISHDETAIL\_WISH FOREIGN KEY (FK\_WISH) REFERENCES WISH (ID\_WISH),

CONSTRAINT FK\_WISHDETAIL\_PRODUCT FOREIGN KEY (FK\_PRODUCT) REFERENCES PRODUCT (ID\_PRODUCT)

);

CREATE TABLE WISH\_HISTORY (

ID\_WISH\_HISTORY SERIAL PRIMARY KEY,

FK\_WISH INT NOT NULL,

WISH\_HISTORY\_DATE DATE NOT NULL,

WISH\_HISTORY\_STATE VARCHAR(10) NOT NULL,

WISH\_HISTORY\_FINAL\_PRICE NUMERIC(12, 2) NOT NULL,

CONSTRAINT FK\_WISHHISTORY\_WISH FOREIGN KEY (FK\_WISH) REFERENCES WISH (ID\_WISH)

);

CREATE TABLE INTERNAL\_USER (

ID\_INTERNAL\_USER SERIAL PRIMARY KEY,

INTERNAL\_USER\_NAME VARCHAR(30) NOT NULL,

INTERNAL\_USER\_PASSWORD VARCHAR(100) NOT NULL,

INTERNAL\_USER\_ROLE VARCHAR(30) NOT NULL

);

CREATE TABLE MAIL (

ID\_MAIL SERIAL PRIMARY KEY,

MAIL\_TYPE VARCHAR(20) NOT NULL,

MAIL\_RECEIVER VARCHAR(100) NOT NULL,

MAIL\_CONTENT VARCHAR(300) NOT NULL

);

INSERT INTO

CLIENT (

CLIENT\_NAME,

CLIENT\_LAST\_NAME,

CLIENT\_EMAIL,

CLIENT\_ADDRESS,

CLIENT\_PASSWORD

)

VALUES

(

'Juan',

'Perez',

'juanperez@gmail.com',

'Larrea 4646',

'123456789'

),

(

'Marta',

'Juana',

'marta@gmail.com',

'Larrea 4645',

'12345678'

),

(

'Martina',

'Morena',

'morena@gmail.com',

'Larrea 4644',

'1234567'

),

(

'Lionel',

'Perez',

'lio@gmail.com',

'Larrea 4643',

'123456'

);

INSERT INTO

PRODUCT (

PRODUCT\_NAME,

PRODUCT\_DESCRIPTION,

PRODUCT\_UNIT\_PRICE,

PRODUCT\_IMAGE

)

VALUES

(

'Paquete a Rio de Janeiro',

'Viaje fantástico para 2',

450000,

'rio\_janeiro.jpg'

),

(

'Paquete a Espana',

'Viaje fantástico para adultos',

550000,

'espana.webp'

),

(

'Paquete a Puerto Rico',

'Hermosas vistas',

700000,

'puerto\_rico.webp'

),

(

'Paquete a Perú',

'Bueno, bonito y barato',

250000,

'peru.jpg'

);

INSERT INTO

WISH (FK\_CLIENT, WISH\_DATE, WISH\_STATE, WISH\_TOTAL)

VALUES

(1, '2025-04-02', 'Pago', 250000),

(2, '2025-05-07', 'Cancelado', 700000),

(3, '2025-01-15', 'Pendiente', 450000);

INSERT INTO

WISH\_DETAIL (

FK\_WISH,

FK\_PRODUCT,

WISH\_DETAIL\_AMOUNT,

WISH\_DETAIL\_SUBTOTAL

)

VALUES

(1, 1, 1, 450000),

(2, 2, 2, 1100000),

(3, 3, 1, 700000);

INSERT INTO

WISH\_HISTORY (

FK\_WISH,

WISH\_HISTORY\_DATE,

WISH\_HISTORY\_STATE,

WISH\_HISTORY\_FINAL\_PRICE

)

VALUES

(1, '2024-12-06', 'Pago', 900000),

(2, '2024-09-06', 'Pago', 700000),

(3, '2024-04-09', 'Cancelado', 500000);

INSERT INTO

INTERNAL\_USER (

INTERNAL\_USER\_NAME,

INTERNAL\_USER\_PASSWORD,

INTERNAL\_USER\_ROLE

)

VALUES

('Martin', 'code12345', 'Agente de viajes'),

('Paula', 'code1234', 'Asesora de ventas'),

('Celina', 'code123', 'Coordinadora de ventas'),

('Ian', 'code12', 'Ejecutivo de ventas');

INSERT INTO

MAIL (MAIL\_TYPE, MAIL\_RECEIVER, MAIL\_CONTENT)

VALUES

(

'CONFIRMACION',

'marta@gmail.com',

'Hola Marta, tu paquete turístico a Perú ha sido reservado exitosamente.'

),

(

'CANCELACION',

'juanperez@gmail.com',

'Hola Juan, tu paquete turístico a Rio de Janeiro ha sido cancelado exitosamente.'

),

(

'CONFIRMACION',

'morena@gmail.com',

'Hola Morena, se ha registrado un inicio de sesión en una Mac 8.'

);

INSERT INTO

WISH\_DETAIL (

FK\_WISH,

FK\_PRODUCT,

WISH\_DETAIL\_AMOUNT,

WISH\_DETAIL\_SUBTOTAL

)

VALUES

(

3,

4,

1,

1 \* (

SELECT

PRODUCT\_UNIT\_PRICE

FROM

PRODUCT

WHERE

ID\_PRODUCT = 4

)

);

UPDATE

WISH\_DETAIL

SET

WISH\_DETAIL\_AMOUNT = 2,

WISH\_DETAIL\_SUBTOTAL = 2 \* (

SELECT

PRODUCT\_UNIT\_PRICE

FROM

PRODUCT

WHERE

ID\_PRODUCT = 4

)

WHERE

FK\_WISH = 3

AND FK\_PRODUCT = 4;

DELETE FROM

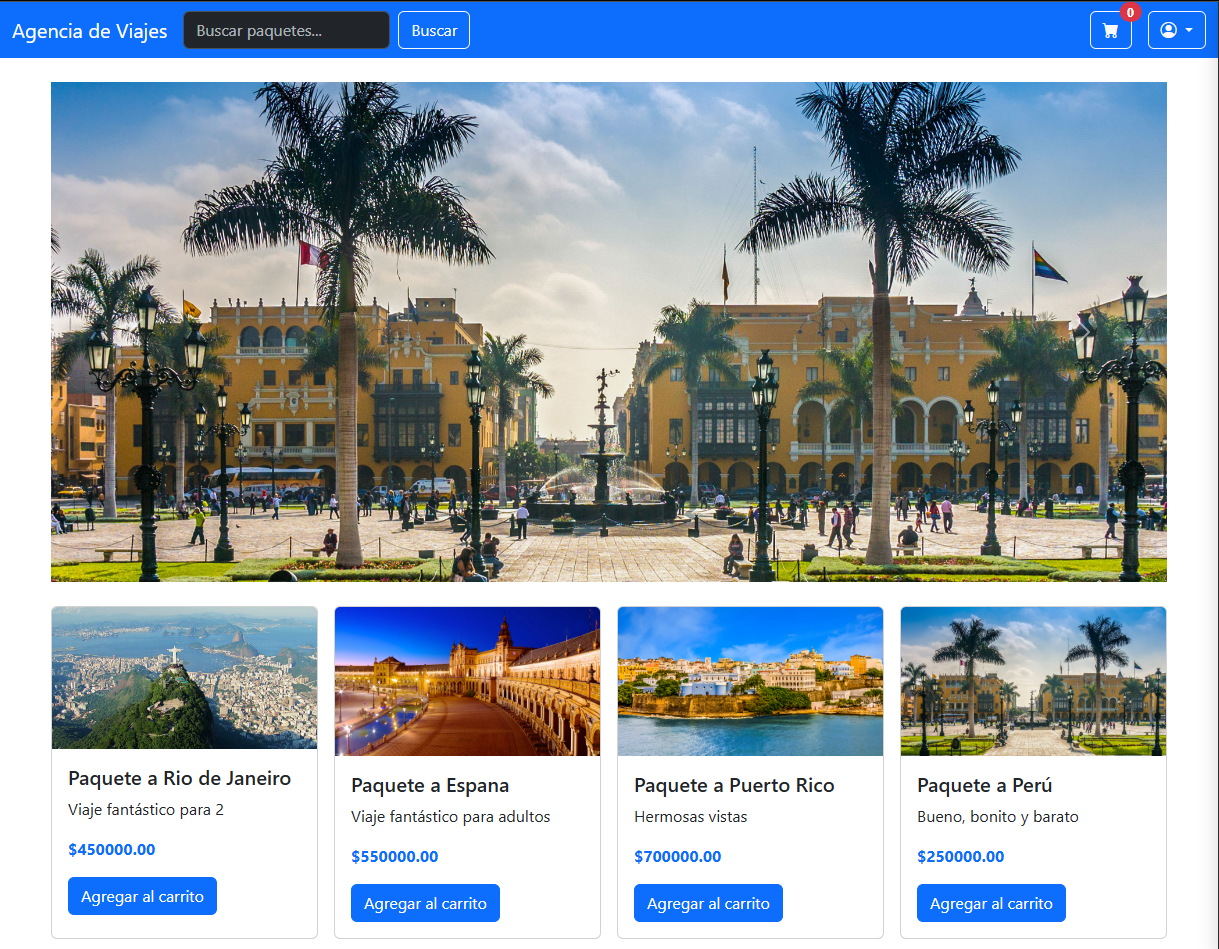
WISH\_DETAIL

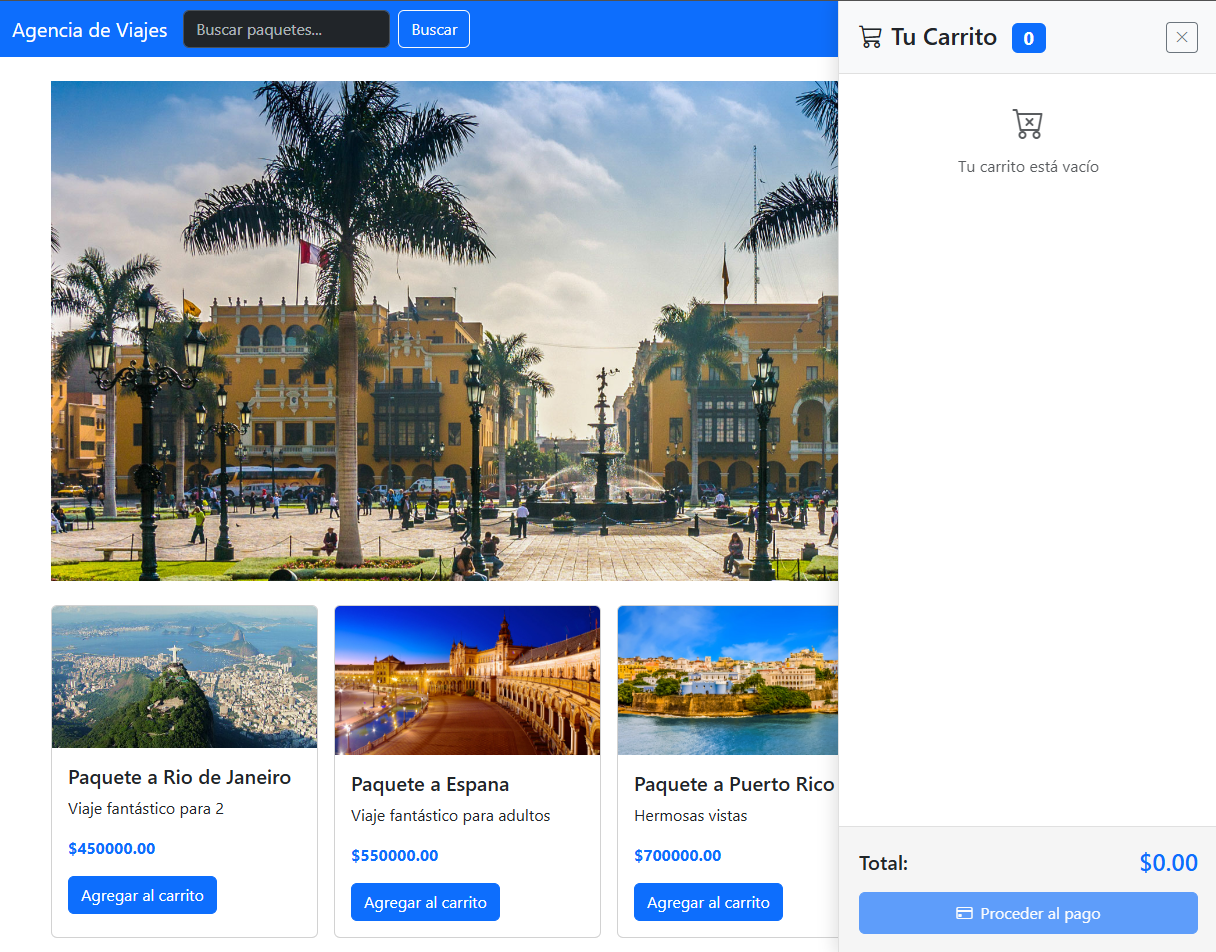
WHERE

FK\_WISH = 3

AND FK\_PRODUCT = 4;

### Vista con paquetes cargados desde base de datos y la función del carrito(vista del cliente):





# 

### Vista de la página de empleado(jefe de ventas):

# 

# 

# 

# Manual de Instalación

## **Requisitos Previos**

## Antes de comenzar, asegúrate de tener instalado:

* 1. **Node.js** (v 15 o superior)
  2. **npm** (Node Package Manager)
  3. **Git** (opcional para clonar el repo)
  4. **MySQL** (o un gestor compatible con el archivo DB.sql)

## **Clonación o descarga del proyecto**

* 1. Descargar el .zip del proyecto y descomprimilo.
  2. El proyecto incluye dos carpetas principales: Backend y Frontend**.**

## **Instalación del Backend**

* 1. Configurar entorno: Abrí una terminal y entrá a la carpeta /Backend. Luego ejecuta: npm install. Por último configura el archivo .env (o crealo si no existe) con las siguientes variables: DB\_HOST=localhost, DB\_USER=tu\_usuario, DB\_PASSWORD=tu\_contraseña, DB\_NAME=nombre\_basedatos
  2. Crear la base de datos: Abrí tu gestor de MySQL y ejecuta el script src/db/DB.sql para crear las tablas necesarias.
  3. Iniciar el servidor
  4. npm start: Esto iniciará el backend por defecto en <http://localhost:3000>.

## **Ejecución del Frontend**

## Entrá en la carpeta Frontend y abrí los archivos HTML directamente en el navegador (doble clic o usar Live Server si usás VS Code).}

# Manual de uso:Cliente

## **Acceso al sitio**

## Ingresá desde cualquier navegador web a la dirección del sitio (por ejemplo: <http://localhost:3000> si estás en entorno local). Se mostrará la página principal con los paquetes turísticos disponibles.

## **Navegación**

## Podés explorar los distintos paquetes turísticos que se muestran dinámicamente desde la base de datos.

## Cada paquete incluye:

* 1. Nombre del destino
  2. Imagen
  3. Precio
  4. Descripción breve
  5. Botón “Agregar al carrito”

## **Carrito de compras**

Al hacer clic en “Agregar al carrito”, el paquete se guarda en tu carrito (desde la base de datos, no localStorage). Podés acceder al carrito desde el ícono o botón “🛒 Carrito”. En el carrito podés:

* 1. Ver los paquetes seleccionados
  2. Eliminar paquetes individuales
  3. Confirmar la compra.

# Manual de Uso: Empleado (Jefe de Ventas)

## **Acceso al sistema**

Para ingresar, el empleado debe acceder a la siguiente dirección: <http://localhost:3000/vista_inicio_emp.html>

Se mostrará un formulario de inicio de sesión que solicita:

1. ID de empleado
2. Contraseña

Actualmente, el formulario está diseñado y presenta validaciones visuales básicas.

Sin embargo, no está conectado al backend, por lo que el login no valida contra la base de datos.

Al presionar el botón de “Iniciar sesión”, el sistema \*\*redirige directamente\*\* al menú del empleado.

## **Menú de gestión de paquetes turísticos**

Una vez dentro, se accede al menú de empleado en: <http://localhost:3000/menu_emp.html>

Desde allí, el jefe de ventas puede gestionar los paquetes turísticos a través de varias secciones:

## **Visualización de paquetes**

La página muestra dinámicamente un listado de paquetes turísticos cargados desde la base de datos.

Cada paquete se presenta con:

1. Nombre del destino
2. Descripción
3. Precio
4. Imagen

Esta es la única funcionalidad que está operativa actualmente y conectada correctamente con el backend y la base de datos.

URL al video demostrativo de las funciones del carrito:

* [Video de Demostración del funcionamiento del carrito](https://drive.google.com/file/d/1WB00TU3-YH_QMFZWIyhYaPrt-g4xJIH7/view?usp=drive_link)

A continuación, dejamos el repositorio con todo el proyecto disponible, con el código detallado arriba y más archivos que no prosperaron:

* <https://github.com/MatiasFavrat/carrito.git>

# Anexo

Anexo – Desglose del Proyecto Web de Venta de Paquetes Turísticos

1. **Frontend** (Interfaz de Usuario)
   1. **Responsable**:Acosta Lionel

**Descripción**:

* + 1. Se encargó del desarrollo visual y estructural del sitio web utilizando HTML, CSS y JavaScript. Creó la página principal, secciones de paquetes turísticos, botones de navegación y formularios. También implementó la lógica básica para interactuar con el backend, como cargar dinámicamente los paquetes y gestionar visualmente el carrito de compras. Se priorizó una estética clara, intuitiva y amigable para el usuario.

1. **Backend** (Servidor y API REST)
   1. **Responsable**:Kettler Thomas

**Descripción**:

* + 1. Desarrolló toda la lógica del servidor utilizando \*\*Node.js con Express\*\*. Implementó rutas para manejar paquetes turísticos, autenticación y el carrito de compras. Cada ruta se conectaba a la base de datos para realizar operaciones CRUD (crear, leer, actualizar, eliminar). También se encargó de estructurar los controladores y middlewares, y asegurar el correcto envío y recepción de datos entre frontend y base de datos.

1. **Base de Datos**
   1. **Responsable**:Favrat Matías

**Descripción**:

* + 1. Diseñó y gestionó la base de datos relacional con PostgreSQL. Creó las tablas necesarias: `usuarios`, `paquetes`, `carrito`, entre otras. Se definieron relaciones, claves primarias y foráneas, y se realizaron consultas SQL para que la información estuviera organizada y disponible para la API. También supervisó la conexión con el backend y realizó pruebas de integridad y rendimiento.

1. **Gestión del Proyecto y Planificación**
   1. **Responsable**: Favrat Matías

**Descripción**:

* + 1. Además de su rol en base de datos, Matías también coordinó las tareas generales del grupo, definiendo cronogramas (Gantt), asignando roles y monitoreando el progreso. Se utilizaron herramientas de planificación básicas como diagramas de tareas y cronogramas semanales.

1. **Funcionalidad del Carrito de Compras**
   1. Trabajo conjunto entre:Kettler Thomas (backend), Favrat Matías (BD), Acosta Lionel (frontend)

Descripción:

* + 1. El carrito se almacena en base de datos, permitiendo persistencia entre sesiones.
    2. Desde el frontend, el usuario puede agregar o eliminar paquetes.
    3. El sistema consulta automáticamente los productos en el carrito y los muestra al usuario en tiempo real.

1. **Pruebas y Validaciones**
   1. **Responsable principal**: Acosta Lionel (interfaz), todo el equipo participó

**Descripción**:

* + 1. Se hicieron pruebas manuales de cada módulo: carga de paquetes, registro de usuario, agregar al carrito y proceso de compra. También se verificó la respuesta del servidor ante errores y el correcto funcionamiento de las rutas.

# Reflexión :

Para comenzar, como grupo decidimos dividir las tareas en función de nuestros intereses y habilidades. Esta organización nos permitió trabajar de forma eficiente desde el principio, combinando momentos de trabajo conjunto con tiempo extraescolar según las responsabilidades asignadas a cada integrante.

Gracias a esta distribución de roles y a una buena organización desde el inicio, no encontramos mayores dificultades en el desarrollo del proyecto. El compañerismo y la comunicación constante fueron fundamentales para mantener un ambiente de trabajo positivo. A pesar de los desafíos que surgieron, podemos afirmar que disfrutamos tanto del proceso como del trabajo en equipo.

Entre los principales inconvenientes que enfrentamos, se destacan la falta de conocimientos específicos en algunas áreas y errores puntuales, como los relacionados con el guardado de paquetes en la base de datos. Sin embargo, logramos resolverlos con investigación, prueba y colaboración.

A nivel personal, coincidimos en que, aunque surgieron diversas dificultades, resultó muy interesante abordar la consigna como si se tratara de un entorno profesional. La forma en que organizamos el trabajo y el nivel de complejidad del proyecto nos permitió vivir una experiencia cercana a la dinámica real de un equipo de desarrollo.

El proyecto logró con éxito la integración entre frontend, backend y base de datos, permitiendo la manipulación de datos en tiempo real y ofreciendo una experiencia funcional de carrito de compras con paquetes turísticos. La aplicación refleja una buena organización del equipo y un enfoque colaborativo durante todas las etapas del desarrollo.

Dado el tiempo disponible y los objetivos planteados para esta instancia, se priorizó la implementación de un prototipo funcional que abarca las funcionalidades principales solicitadas en la consigna.

Algunas funciones no pudieron concretarse en esta etapa (como el añadir/edición/eliminar de los paquetes,el login/registro de los clientes y login de los empleados), aunque sí fueron tenidas en cuenta dentro del diseño general de la arquitectura.

El prototipo entregado cumple con los objetivos mínimos del proyecto y sienta las bases para una futura evolución hacia una aplicación completa, incorporando las funcionalidades restantes y mejorando el rendimiento y la seguridad.

# Bibliografía:

Bharti, S. (2024, November 23). Exploring NGINX: a robust web server and reverse proxy. *Medium*. <https://blog.devops.dev/exploring-nginx-a-robust-web-server-and-reverse-proxy-fe1e33ea5749>

Bukhari, S. M. K. (2024, May 28). *A comprehensive guide to NPM (Node Package Manager)*. DEV Community. <https://dev.to/smkbukhari/a-comprehensive-guide-to-npm-node-package-manager-30n4>

Instaclustr. (2025, April 28). *Complete guide to PostgreSQL: Features, use cases, and tutorial*. <https://www.instaclustr.com/education/postgresql/complete-guide-to-postgresql-features-use-cases-and-tutorial/>

SSL Corp. (2023, December 8). *What is SSL/TLS: An In-Depth Guide - SSL.com*. SSL.com. <https://www.ssl.com/article/what-is-ssl-tls-an-in-depth-guide/>

Solanki, M. (2025, May 22). *Standard SSL certificate explained: Domain-Level HTTPS Security*. SSL2BUY. <https://www.ssl2buy.com/wiki/standard-ssl-certificate-guide>

*Ubuntu Server documentation*. (n.d.). Ubuntu Server. <https://documentation.ubuntu.com/server/>

*Index | Node.js v24.2.0 Documentation*. (n.d.). <https://nodejs.org/docs/latest/api/>

*W3Schools.com*. (n.d.). <https://www.w3schools.com/html/html_css.asp>

*W3Schools.com*. (n.d.-b). <https://www.w3schools.com/postgresql/postgresql_intro.php>

*W3Schools.com*. (n.d.-c). <https://www.w3schools.com/js/js_intro.asp>