Documentación de Analíticas de price-match y abandono de carrito en reservas

Nombre y Apellidos: Eduardo Sebastián Dávila Zevallos
Programa Formativo: Certificado de Profesionalidad en Confección y Publicación de Páginas Web (IFCD0110)
Centro de Formación: I.E.S. Bárbara de Braganza, Badajoz
Empresa de Prácticas: THB Hotels

Índice

1. Descripción de la página

El proyecto consiste en el desarrollo de una página enfocada al análisis de reservas que presentan price-match (comparación de precios con OTAs como Booking, Expedia y otras agencias de viaje online) y reservas que presentaron abandono de carrito, es decir, usuarios que iniciaron el proceso de reserva pero lo abandonaron antes de completarlo. La página contiene filtros para cambiar de una analítica a otra, para cambiar el tipo de rango de fecha (por fecha de creación de reserva o de check-in/check-out) y contiene botones para ajustar el rango de fechas.

El gráfico representa el total de reservas, las reservas con price-match o abandono de carrito, y de esas cuentas fueron finalizadas y canceladas.

La página también contiene un buscador por código de soporte y localizador, que devuelve los datos de esa reserva y si tiene price-match y/o presenta abandono de carrito.

El módulo ha sido desarrollado sobre el ecosistema Laravel utilizando JavaScript moderno basado en el estándar ES6, jQuery para la manipulación del DOM y DataTables para la explotación de datos analíticos.

2. Objetivos

Optimizar la conversión: Proveer visibilidad sobre cuántas reservas se completan o cancelan tras interactuar con ventajas competitivas (Price-Match) o incentivos de recuperación (Reminder Popup: ventana emergente mostrada al usuario cuando abandona el proceso de reserva con el objetivo de incentivar su finalización).

Centralizar la auditoría de reservas: Facilitar al equipo de soporte un motor de búsqueda rápido e inteligente que unifique el estado de las métricas analíticas adjuntas a un localizador.

Flexibilidad Temporal: Permitir el análisis cruzado evaluando tanto la salud financiera en el momento de la compra (Fecha de creación) como la ocupación hotelera real (Fecha de estancia).

3. Funcionalidades

Gestión de Filtros Dinámicos

- Conmutador de Estadísticas: Permite cambiar el contexto entre Price-Match y Abandono de Carrito sin recargar la página. Al cambiar el tipo:
    - Se reajustan los anchos de los contenedores (70% vs 85%).
    - Se ocultan o muestran componentes de cálculo intermedios como el formulario de datos del cliente (#total-with-hotel).
    - Se mutan los textos de las tablas y títulos automáticamente.
- Modo de Rango de Fecha: Alterna entre filtrar por creación ("Booking") o por estancia ("Stay").
- Reglas de Validación de Fechas: * Si el filtro es "Por fecha de creación", el sistema restringe el calendario bloqueando fechas futuras (max, today).
    - Control de colisiones: Si la fecha inicial es superior a la final, invoca los selectores nativos (showPicker()) para forzar la corrección del rango.

Gráficos Proporcionales Dinámicos (StatsBar)

Un componente desarrollado en JavaScript que renderiza barras horizontales continuas mediante variables CSS dinámicas (--height, --previous, --next).

- Representa de forma nativa la composición volumétrica del embudo de conversión (Totales -> Segmento Analizado -> Finalizados / Cancelados).
- Calcula porcentajes reales dinámicamente frente a su base de datos correspondiente en Backend.

Buscador Avanzado de Reservas

- Posee un algoritmo de validación previa en Frontend mediante expresiones regulares (/^[A-Z0-9]+$/).
- Determina la estrategia de consulta según la longitud del código introducido:
    - 6 caracteres: Realiza una búsqueda por código de soporte (Token).
    - 9 caracteres: Realiza una búsqueda por localizador de reserva.
- Muestra un spinner de carga asíncrono y renderiza el desglose de habitaciones, adultos y niños asociados en caso de éxito.

Tabla de Explotación de Datos

- Integración con DataTables 2.3.7.

- Inclusión de herramientas nativas para la exportación y descarga de informes analíticos en formatos Excel (JSZip), PDF (PdfMake) e impresión directa.

4. Descripción General de las Páginas

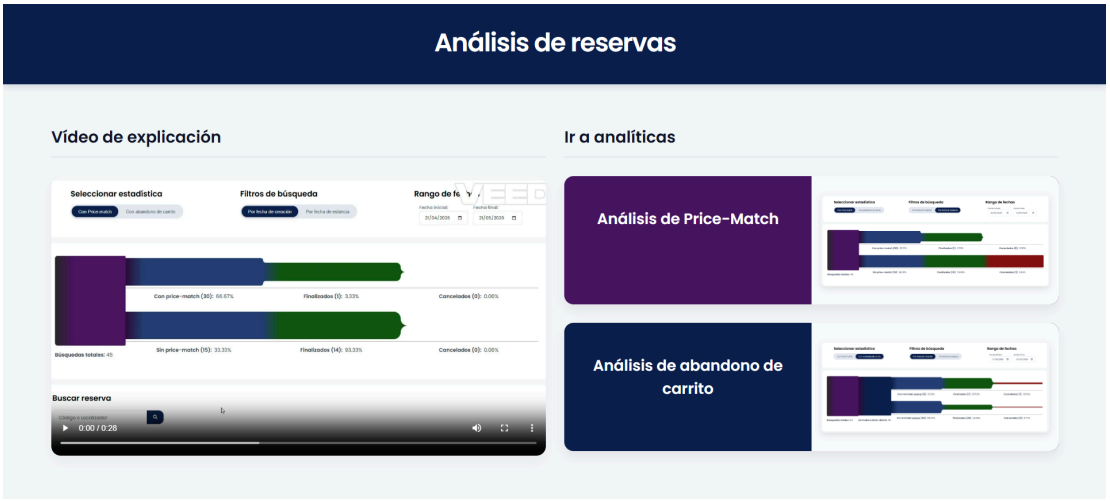
El sistema se compone de dos interfaces principales integradas en la estructura de plantillas de la aplicación:

A. Vista Principal (Home)

Funciona como el panel de bienvenida y distribución. Ofrece material de inducción (reproductor de vídeo explicativo) y dos accesos directos (Cards) que redirigen a los flujos de negocio específicos inyectando parámetros de configuración por defecto:

Análisis de Price-Match: Comparación de tarifas entre la web oficial de THB Hotels y OTAs como Booking, Expedia y otras agencias de viaje online.

Análisis de Abandono de Carrito: métricas relacionadas con usuarios que iniciaron una reserva pero no finalizaron el proceso de compra.



B. Vista de analíticas

Un panel dinámico e interactivo (Dashboard) que permite segmentar y auditar el comportamiento de las reservas. Modifica sus textos, el comportamiento de sus gráficos de barras proporcionales y las peticiones asíncronas en tiempo real según los filtros activos seleccionados por el usuario.

Buscador de reservas: El componente Buscador Avanzado de Reservas es una herramienta analítica y de auditoría integrada en el dashboard. El buscador incluye alertas visuales por colores para representar si tiene price-match y/o si el usuario interactuó con la ventana emergente de abandono de carrito.

Entre los datos se incluye: Identificación y Control (localizadores y cupones), detalles del Hotel, desglose de estancia, fecha de reserva, check-in, check-out y datos del cliente (nombre, email, teléfono y dirección). De este modo, el buscador centraliza la información crítica y los indicadores de conversión del negocio en un único impacto visual directo.

5. Arquitectura del Sistema

El módulo sigue un flujo asíncrono desacoplado en tres capas lógicas (Modelo-Vista-Controlador + JavaScript moderno, jQuery y DataTables). La comunicación se realiza mediante peticiones HTTP asíncronas (AJAX JSON) que recalculan las métricas y actualizan el estado de la interfaz de forma dinámica y sin refrescos de página.

A. Capa de Presentación (Frontend)

La lógica del cliente está encapsulada en un objeto modular único BookingAnalytics que actúa como el orquestador o Mediador de la interfaz, gestionando eventos, el componente de gráficos customizados (StatsBar) y las peticiones asíncronas de datos.

- **Manejador de estado centralizado**: Mediante los métodos init() y handleFilterClick(), el script sincroniza el tipo de análisis (price-match o reminder-popup) mutando dinámicamente anchos de contenedores CSS alterando la visibilidad de formularios intermedios (#total-with-hotel) y modificando las etiquetas de texto en las cabeceras de las DataTables.
- **Validaciones de rango de fechas**: El método applyDateTypeRules() valida de forma nativa si el filtrado es por creación (booking), inyectando el atributo HTML max para bloquear la selección de fechas futuras.
- **Control de colisiones**: Si la fecha inicial supera a la final, el método handleDateChange() frena la petición asíncrona e invoca de manera forzada el selector nativo del navegador (.showPicker()) en el input erróneo para guiar al usuario.
- **Buscador**: El método runSearch() valida el formato del string introducido mediante la expresión regular /^[A-Z0-9]+$/ en mayúsculas y discrimina su longitud (6 para *Token de Soporte* o 9 para *Localizador de Dingus*) antes de disparar la consulta POST al servidor.

Funcionamiento Interno del Componente StatsBar

StatsBar es un componente reutilizable desarrollado en JavaScript para representar embudos de conversión mediante barras proporcionales dinámicas. Su diseño desacoplado permite reutilizarlo en futuros desarrollos analíticos de THB Hotels sin depender de librerías externas.

1. **Instanciación Configurable**: La Factory interna del frontend getStatsConfig() genera dos instancias en memoria (pm y noPm), mapeando las claves del JSON esperado (pm_all, pm_completed, pm_cancelled) junto a sus colores corporativos y sus multiplicadores visuales.
2. **Cálculo de variables dinámicas**: El método update() itera recursivamente sobre los pasos configurados. Si el elemento posee una propiedad base, calcula el porcentaje matemático real ((rawValue / baseVal) * 100).

3. **Pintado mediante propiedades CSS Reactivas**: El método setStat() inyecta los valores calculados directamente en las variables personalizadas del CSS (--height, --previous, --next). El archivo CSS utiliza estas variables para ejecutar cálculos matemáticos en tiempo real y aplicar bordes redondeados y degradados con transiciones fluidas de forma nativa.

Integración y Consumo con AJAX

La capa de frontend utiliza jQuery AJAX para conectarse con los endpoints expuestos por Laravel:

- **Carga de estadísticas (GET /get-price-match-analytics)**: Envía las fechas serializadas (start_date, end_date), el tipo de rango (date_type) y el tipo de analítica (analysis_type). Al recibir el JSON exitoso, desactiva el *loader* visual y al recibir el JSON exitoso, desactiva el loader visual y ejecuta el método update() de las barras.
- **Búsqueda unificada (POST /search-booking-by-token-or-localizer)**: Envía el código limpio. En caso de éxito, decodifica las colecciones JSON complejas en el cliente (como el mapeo indexado en bucle de rooms y rooms_search para mostrar adultos, niños, tarifas y códigos de habitación) y procesa los datos financieros utilizando la API internacional Intl.NumberFormat('es-ES') para formatear la moneda en euros.

B. Capa de Control (Laravel Controller)

El controlador BookingAnalyticsController actúa como la API de control de accesos e intermediario de peticiones del módulo. Su responsabilidad es la sanitización de parámetros de entrada, el control de errores de formato temporal y el retorno de respuestas estructuradas.

Listado de Implementaciones:

- **Protección contra inyecciones y arrays maliciosos**: Antes de parsear las fechas, el controlador verifica mediante is_array() que los parámetros del request no hayan sido alterados maliciosamente en la petición HTTP, retornando arrays vacíos o respuestas JSON inmediatas si detecta anomalías.
- **Captura de excepciones temporales**: Utiliza la librería Carbon::parse() dentro de bloques try-catch capturando específicamente la excepción InvalidFormatException en caso de que las cadenas de texto de las fechas hayan sido manipuladas con formatos inválidos.

- **Lógica enrutada por longitud (*Match Expression*)**: En el método searchBookingByTokenOrLocalizer(), se explota la estructura condicional moderna match de PHP según la longitud del código (strlen), realizando una validación gemela de seguridad con ctype_alnum() y strtoupper() antes de invocar la consulta específica al modelo.

C. Capa de Datos (Modelos Eloquent / Database)

El modelo Booking representa la abstracción de la base de datos de reservas. Al trabajar con columnas complejas en formato JSON, centraliza la deserialización de campos y optimiza las consultas agregadas directamente a través de sentencias a nivel de base de datos.

Listado de Implementaciones:

- **Mapeo de atributos**: El método nativo casts() convierte automáticamente cadenas de texto JSON de la base de datos en colecciones y arrays nativos de PHP para campos críticos como price_match, customer_data, payment_data, rooms y rooms_search.
- **Lógica SQL (priceMatchAnalytics)**: En lugar de traer miles de registros a la memoria de PHP para contarlos, el modelo inyecta un bloque de consultas optimizado con selectRaw(). Al discriminar el contexto (analysis_type), utiliza funciones nativas como SUM() y COUNT() evaluando operaciones condicionales lógicas integradas diretamente dentro del motor de base de datos:

  SUM(hotel IS NOT NULL AND reminder_popup_shown = 1 AND confirmation_email_sent = 1) AS pm_completed

- **Estrategias de segmentación temporal fluida**: El modelo altera programáticamente sus cláusulas de rango. Si el parámetro es stay, aplica condiciones de cruce hotelero (where('check_in', '<=', $endDate)->where('check_out', '>=', $startDate)), mientras que si es booking, utiliza un rango indexado estándar sobre la marca de tiempo de creación (whereBetween('created_at')).

6. Despliegue a Producción (Hosting IONOS)

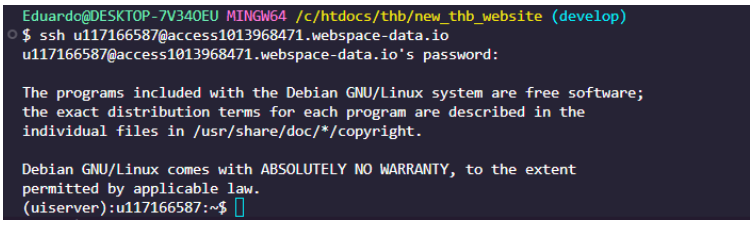
Este apartado describe el procedimiento estándar para publicar la aplicación new_thb_website en el entorno de producción (servidor compartido de IONOS). El objetivo de esta guía es garantizar un despliegue seguro, minimizando el tiempo de inactividad del sitio y previniendo los errores de configuración comunes derivados de las restricciones del hosting.

Requisitos Previos

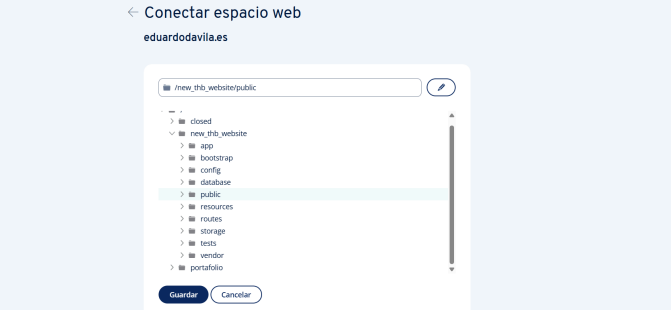
- Acceso al Panel de Control de IONOS.
- Credenciales de acceso SSH (u117166587).
- Base de datos MySQL creada en el hosting y credenciales anotadas.

Secuencia de configuración para realizar el despliegue:

1. **Preparación del Entorno (Conexión SSH)**: Me he conectado usando mi usuario y contraseña asignadas por el proveedor.



2. **Carga y Descompresión del Proyecto**: A través del panel de control del proveedor he introducido el archivo comprimido .zip del proyecto y mediante consola lo he descomprimido dentro del hosting.



3. **Enrutamiento del Dominio**: Dentro del panel de control he ajustado el destino para que apunte a public de mi proyecto, ya que contiene el index.php.

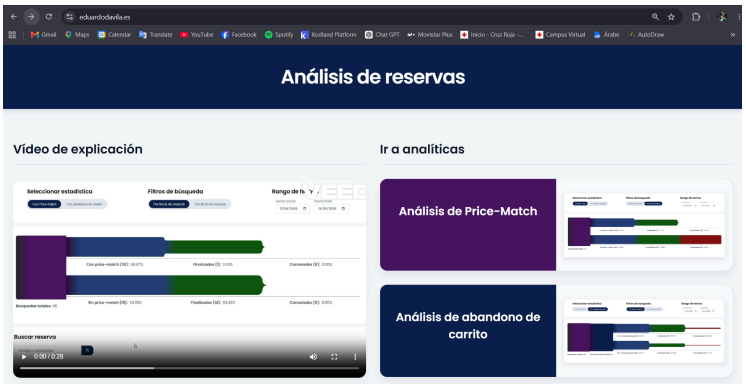
4. **Configuración del Entorno (.env)**: Dentro del hosting, he configurado la carpeta .env con

```
 GNU nano 5.4                                                    .env *
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:BCeAbUB3AUueS0oCFz3p95qpmNZiVQKK0/07Pag1RQA=
APP_DEBUG=true
APP_URL=https://eduardodavila.es/new_thb_website/

APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US

BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_LEVEL=debug
```

5. **Optimización de Memoria y Caché**: Al configurar el .env, he limpiado la caché de las vistas y la configuración.

```
(uiserver):u117166587:~$ cd new_thb_website
(uiserver):u117166587:~/new_thb_website$ php artisan view:clear
Content-type: text/html

(uiserver):u117166587:~/new_thb_website$ php artisan config:clear
Content-type: text/html

(uiserver):u117166587:~/new_thb_website$ php artisan cache:clear
Content-type: text/html

(uiserver):u117166587:~/new_thb_website$ chmod 644 *.blade.php
chmod: cannot access '*.blade.php': No such file or directory
(uiserver):u117166587:~/new_thb_website$ find . -type d -exec chmod 755 {} \;
(uiserver):u117166587:~/new_thb_website$ find . -type f -exec chmod 644 {} \;
(uiserver):u117166587:~/new_thb_website$ /usr/bin/php8.4 artisan view:clear

  INFO  Compiled views cleared successfully.

Content-type: text/html; charset=UTF-8

(uiserver):u117166587:~/new_thb_website$
```

6. **Migraciones y Asignación de Permisos**: Luego de terminar de realizar las migraciones y dar los permisos necesarios a mis carpetas ya se puede visualizar la web en el hosting.



7. Gestión del Control de Versiones y Flujo de Trabajo (Git & Gitea)

Para asegurar la trazabilidad del código, el desarrollo de este módulo se ha gestionado íntegramente bajo el sistema de control de versiones Git, utilizando la plataforma interna Gitea de THB Hotels como repositorio remoto.

A. Infraestructura y Acceso Seguro

Debido a las políticas de seguridad de la compañía y al carácter privado del código fuente del proyecto, el acceso al ecosistema de desarrollo requiere autenticación previa en el acceso mediante VPN y credenciales.

B. Ciclo de comandos utilizados en el proyecto

El flujo de trabajo diario y la preparación del entorno se documentan a través de los siguientes comandos nativos de Git ejecutados en la consola:

Inicialización y Vinculación del Repositorio: Para transformar el directorio local del proyecto Laravel en un repositorio Git y conectarlo con el servidor de la empresa, se emplearon los comandos:

```
git init
git remote add origin http://sphere.thbhotels.com:3000/thb-formacion/price-match-analytics
```

Gestión de Ramas: se estableció una estrategia basada en tres niveles de ramas:

- main: versión estable en producción.
- develop: integración de funcionalidades pendientes de publicación.
- feature/*: desarrollo individual de cada *nueva funcionalidad*.

Cada nueva funcionalidad desarrollada durante las prácticas fue implementada inicialmente en una rama feature independiente antes de ser integrada en la rama de desarrollo (develop), esta nueva rama de feature debe ser creada a partir de main, que es la rama principal y la más estable.

```
# Creación de una nueva rama a partir de la rama main (la más estable)
git checkout -b new-feature
```

Confirmación de cambios: Cada vez que se completaba un bloque funcional del código en la rama de feature, se guardaba el estado del código realizando un commit:

```
git add .
git commit -m "Fix booking search filters"  # Registro del cambio realizado
```

Sincronización con el servidor remoto:

```
# Sincronización de la rama con el servidor remoto
git push origin new-feature
```

C. Flujo de Integración: Pull Requests (PR) en Gitea

Una vez que una nueva funcionalidad (feature) se encuentra completamente desarrollada y testeada en el entorno local, el flujo de trabajo de la empresa exige la apertura de una Pull Request (PR) a través de la interfaz web de Gitea para iniciar el proceso de integración en el repositorio central.

Petición de fusión: El ciclo de vida del código sigue un camino estricto de promoción a través de las ramas principales:

- **De feature a develop**: Tras completar una tarea, se solicita la fusión de la rama de la funcionalidad hacia develop. Aquí es donde se consolida el trabajo diario y se realizan las pruebas de integración.

- **De develop a main**: Una vez que el conjunto de funcionalidades en develop ha sido validado, testeado y se considera estable, se abre una nueva Pull Request hacia main para publicar una nueva versión oficial en el entorno de producción.

Auditoría de Código: Antes de aprobar la fusión, otros desarrolladores del equipo realizan una revisión del código:

- **Rendimiento e impacto en BD**: Se analiza el comportamiento de las consultas SQL generadas por el ORM Eloquent de Laravel. El objetivo es detectar problemas comunes, evaluar los tiempos de ejecución y plantear optimizaciones mediante índices o optimización de queries.
- **Calidad y sintaxis**: Se comprueba que el código se adhiere a los estándares de Laravel y a las buenas prácticas de desarrollo.
- **Mantenibilidad**: Se revisa la limpieza general del código para asegurar que sea legible y escalable para el resto del equipo.

8. Optimización para buscadores (SEO) y estructura de metadatos

En cumplimiento con los estándares de publicación web, se ha dotado a la página principal del módulo (Home) de una estructura de metadatos optimizada para buscadores y redes sociales.

A. Descriptores y metadatos técnicos implementados

Los descriptores (meta-etiquetas) sintetizan el contenido de la página para los motores de búsqueda. En la plantilla de la vista principal se han inyectado los siguientes bloques en el <head> del HTML:

```
<title>Analíticas de price-match y abandono de carrito | THB Hotels</title>
<meta name="description" content="Panel interno de auditoría para el análisis de Price-Match de THB Hotels.">
<meta name="keywords" content="thb hotels, analitica web, price match, abandono carrito, auditoria reservas, panel interno">
<meta name="author" content="Eduardo Sebastián Dávila Zevallos">
```

B. Integración de gráficos abiertos (Open Graph) y Tarjetas de redes sociales

Para asegurar que la página se renderiza de forma atractiva con su título, descripción e imágenes corporativas en caso de ser compartida en entornos profesionales o plataformas de comunicación interna, se han configurado los protocolos de Open Graph (Facebook) y Twitter Cards:

```
<meta property="og:type" content="website">
<meta property="og:title" content="Dashboard de Analíticas y Optimización de Reservas | THB Hotels">
<meta property="og:description" content="Panel interno de auditoría para el análisis de Price-Match y recuperación de abandono de carrito.">
<meta property="og:url" content="http://thb-formacion/price-match-analytics">
<meta property="og:image" content="http://thbhotels.com/images/seo/og-analytics-preview.png">
<meta property="og:image:width" content="1200">
<meta property="og:image:height" content="630">

<meta name="twitter:card" content="summary_large_image">
<meta name="twitter:title" content="Dashboard de Analíticas y Optimización de Reservas | THB Hotels">
<meta name="twitter:description" content="Panel interno de auditoría analítica de THB Hotels.">
<meta name="twitter:image" content="http://thbhotels.com/images/seo/og-analytics-preview.png">
```

C. Optimización para dispositivos Apple (iOS)

Para mejorar la experiencia de usuario (UX) en movilidad y permitir que los miembros del equipo guarden la herramienta en la pantalla de inicio de sus dispositivos Apple como si fuera una aplicación nativa (PWA / Web App), se incorporaron las etiquetas e iconos específicos:

```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
<meta name="apple-mobile-web-app-title" content="THB Analytics">
<link rel="apple-touch-icon" sizes="180x180" href="/images/seo/apple-touch-icon.png">
```

9. Conclusiones y Valoración de Competencias

La realización de este proyecto en el entorno real de THB Hotels ha supuesto la consolidación práctica de los conocimientos teóricos adquiridos a lo largo del programa formativo del Certificado de Profesionalidad en Confección y Publicación de Páginas Web (IFCD0110). El diseño, desarrollo e implementación del módulo analítico de reservas ha permitido validar las siguientes competencias clave:

- **Consolidación técnica y arquitectura de software**: El proyecto ha permitido dar el salto desde la programación estructurada básica hacia arquitecturas desacopladas complejas. La correcta separación de responsabilidades bajo el patrón MVC en Laravel en el Backend, sumada al desarrollo de funcionalidades dinámicas mediante JavaScript moderno, demuestra la capacidad de desarrollar software escalable, mantenible y eficiente según los estándares actuales del sector.

- **Optimización de recursos y rendimiento**: Uno de los mayores aprendizajes ha sido comprender a gestionar el rendimiento. El uso de técnicas como el casting en los modelos Eloquent, la delegación de operaciones lógicas al motor de base de datos a través de selectRaw(), el uso de índices para mejorar el rendimiento de las consultas y la manipulación precisa del DOM con jQuery y DataTables garantizan una experiencia de usuario fluida y un consumo menor de recursos en el servidor.