Prototipo de plataforma móvil y e-wallet para la digitalización y promoción de restaurantes. 197/253.

Prototype of a mobile platform and e-wallet for the digitalization and promotion of restaurants. 197/253.

Pablo Jesús Gómez Pérez (1).
Estudiante Tecnológico Nacional de México/I. T. de Tuxtla Gutiérrez.
L21270151@tuxtla.tecnm.mx.

María Guadalupe Monjaras Velasco* (2). Tecnológico Nacional de México/Instituto Tecnológico de Tuxtla Gutiérrez, maria.mv@tuxtla.tecnm.mx.

Imelda Valles López (3), Tecnológico Nacional de México/Instituto Tecnológico de Tuxtla Gutiérrez, imelda.vl@tuxtla.tecnm.mx.

Elfer Isaías Clemente Camacho (4), Tecnológico Nacional de México/Instituto Tecnológico de Tuxtla Gutiérrez, elfer.cc@tuxtla.tecnm.mx.

Ciclalli Cabrera García (5), Tecnológico Nacional de México/Instituto Tecnológico de Tuxtla Gutiérrez, ciclalli.cg@tuxtla.tecnm.mx.

Raúl Parades Trinidad (6), Tecnológico Nacional de México/Instituto Tecnológico de Tuxtla Gutiérrez, raul.pt@tuxtla.tecnm.mx.

*corresponding author.

Artículo recibido en mayo 25, 2026; aceptado en junio 26, 2026.

Resumen.

Este artículo describe el desarrollo de un prototipo que consiste en una solución tecnológica integral para impulsar la transformación digital y la competitividad de los establecimientos de convivencia social con servicios de bebidas y cocina típica en Tuxtla Gutiérrez, Chiapas. El proyecto se centra en la implementación de una arquitectura escalable que incluye una aplicación móvil y un monedero electrónico dedicados a la promoción y fidelización de clientes. esta plataforma se compone de un backend robusto, desarrollado sobre tecnologías de desarrollo web y móvil modernas, y un panel administrativo que permite a los propietarios la autogestión de contenidos, promociones y programas de lealtad. La aplicación móvil utiliza técnicas de personalización y filtrado avanzado para ofrecer recomendaciones relevantes a los usuarios, incentivando la exploración de nuevos negocios. Se introduce el e-wallet como mecanismo de transacción y recompensa, midiendo su impacto en la lealtad del cliente y la economía local.

Palabras claves: Aplicación móvil, backend robusto, JSON, maps API, monedero electrónico.

Abstract.

This article describes the development of a prototype that consists of a comprehensive technological solution to promote digital transformation and the competitiveness of social coexistence establishments with beverage and typical cuisine services in Tuxtla Gutiérrez, Chiapas. The project focuses on the implementation of a scalable architecture that includes a mobile application and an electronic wallet dedicated to customer promotion and loyalty. This platform consists of a robust backend, developed on modern web and mobile development technologies, and an administrative panel that allows owners to self-manage content, promotions and loyalty programs. The mobile application uses customization techniques and advanced filtering to offer relevant recommendations to users, encouraging the exploration of new businesses. The e-wallet is introduced as a transaction and reward mechanism, measuring its impact on customer loyalty and the local economy.

Keywords: Mobile application, robust backend, JSON, Maps API, e-wallet.

1. Introducción.

La industria porcina se considera una de las más aceptadas a nivel mundial gracias al significativo consumo de carne de este tipo de animales, tan solo en 2019, el volumen de producción de carne de porcino en México superó los 1,6 millones de toneladas métricas. Esto representó un incremento de alrededor del 6,6% en comparación con el volumen de producción reportado un año anterior.

En Chiapas, y particularmente en Tuxtla Gutiérrez, los establecimientos de alimentos y bebidas, especialmente los establecimientos botaneros, desempeñan un papel fundamental en la cultura y la economía local. Estos lugares no solo son puntos de encuentro social, sino que también representan una expresión de la identidad chiapaneca, donde se fusionan tradiciones culinarias, música y convivencia. Sin embargo, a pesar de su relevancia cultural, muchos de estos negocios enfrentan desafíos significativos para mantenerse competitivos en un mercado cada vez más exigente.

De acuerdo con datos del Instituto Nacional de Estadística y Geografía (INEGI, 2023), el sector de restaurantes y bares en Chiapas contribuye con aproximadamente el 8% del PIB estatal, y Tuxtla Gutiérrez concentra el 40% de estos establecimientos en la región. No obstante, la competencia es alta, y muchos negocios, especialmente los pequeños y medianos, carecen de herramientas efectivas para promocionarse y fidelizar a sus clientes. Esta situación se ve agravada por la falta de acceso a tecnologías digitales avanzadas, lo que limita su capacidad para atraer a un público más amplio y adaptarse a las demandas del consumidor moderno.

Los centros botaneros, en particular, son emblemáticos de la cultura chiapaneca. Según un estudio de la Secretaría de Turismo del Estado de Chiapas (2022), estos establecimientos son visitados por el 70% de los turistas que llegan a Tuxtla Gutiérrez, ya que ofrecen una experiencia única que combina gastronomía local, música en vivo y un ambiente festivo. Sin embargo, muchos de estos negocios no cuentan con estrategias digitales efectivas para promocionarse, lo que reduce su visibilidad y su capacidad para atraer tanto a turistas como a clientes locales. Además, los consumidores en Tuxtla Gutiérrez buscan experiencias más personalizadas y promociones atractivas, pero no siempre encuentran opciones que se ajusten a sus preferencias. Una encuesta realizada por la Cámara Nacional de la Industria de Restaurantes y Alimentos Condimentados (CANIRAC, 2023) reveló que el 65% de los clientes en la ciudad prefieren establecimientos que ofrezcan descuentos o sistemas de recompensas, mientras que el 55% valora la posibilidad de descubrir nuevos lugares a través de plataformas digitales. Sin embargo, no existe una solución integral que combine la exploración de establecimientos con un sistema de fidelización que incentive la visita a múltiples lugares. Por otro lado, un estudio de Google México (2023) reportó que el 80% de los consumidores en México utilizan sus dispositivos móviles para buscar restaurantes, bares y otros establecimientos de entretenimiento, mientras que el 60% toma decisiones basadas en reseñas y recomendaciones en línea. Este dato subraya la importancia de contar con una presencia digital sólida, algo que muchos establecimientos en Tuxtla Gutiérrez aún no han logrado consolidar. Para fortalecer la justificación del presente estudio, es importante destacar los principales beneficios que el uso de tecnologías digitales puede aportar a los establecimientos de alimentos y bebidas, en particular a los centros botaneros. La digitalización de estos negocios no solo contribuye a su promoción y crecimiento, sino que también les permite

mejorar la experiencia de sus clientes mediante estrategias de fidelización, personalización de servicios y análisis de datos.

Desde el punto de vista académico, el desarrollo de este sistema permitirá aplicar y consolidar conocimientos en arquitectura de software, desarrollo backend con Spring Boot y Java, gestión de bases de datos relacionales, diseño de APIs RESTful, desarrollo móvil con Flutter, y diseño de interfaces web con Angular. Asimismo, se desarrollarán habilidades en planificación de proyectos, trabajo colaborativo y análisis de requerimientos reales de negocio.

Este proyecto no solo tiene un impacto tecnológico y académico, sino también un valor económico y social, al impulsar la digitalización de los negocios locales, promover la economía regional y ofrecer una herramienta útil tanto para empresarios como para ciudadanos.

2. Métodos.

Se describen los procedimientos considerados más significativos en el diseño, desarrollo, documentación y en la implementación de la seguridad del proyecto:

Diseño.

Los patrones de diseño aplicados son inyección de dependencias y arquitectura Service-Repository-Controller-Model. En el desarrollo del backend del proyecto 300 Lugares, se implementaron patrones de diseño ampliamente utilizados en la ingeniería de software moderna, especialmente en aplicaciones construidas con el framework Spring Boot. Entre estos patrones, destacan la inyección de dependencias y la arquitectura Service-Repository-Controller-Model, que juntos permiten un diseño desacoplado, limpio y mantenible.

Inyección de dependencias (Dependency Injection).

De acuerdo con Gamma et al. (1995), uno de los principios más importantes en el diseño de software es la inversión de dependencias, el cual promueve que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino que ambos deben depender de abstracciones. La inyección de dependencias (DI) es una forma concreta de aplicar este principio.

Según la documentación oficial de Spring Framework (Spring, 2024), la inyección de dependencias consiste en proporcionar los objetos requeridos (dependencias) a una clase en tiempo de ejecución, en lugar de que la clase los cree directamente. Esto permite un bajo acoplamiento entre componentes, favorece la reutilización de código y facilita las pruebas unitarias.

Arquitectura Service-Repository-Controller-Model.

Este patrón de capas distribuye las responsabilidades del sistema de manera modular, siguiendo principios SOLID. De acuerdo con Alur et al. (2003), separar las preocupaciones en distintos niveles mejora la escalabilidad del sistema y facilita su mantenimiento.

La arquitectura aplicada en este proyecto consta de las siguientes capas:

- Model (Entidad/DTO): Representa las estructuras de datos y entidades que reflejan las tablas de la base de datos. Utilizan anotaciones de JPA como @Entity, @Id, @ManyToOne, etc. Además, se emplearon DTOs (Data Transfer Objects) para encapsular y transportar datos entre capas, evitando exponer directamente las entidades.
- Repository: Contiene interfaces que extienden de JpaRepository o CrudRepository de Spring Data JPA. Esta capa se encarga de las operaciones CRUD y consultas derivadas a la base de datos. Se mantiene completamente separada de la lógica de negocio.

- Service: Encapsula la lógica de negocio. Aquí se procesan los datos, se llaman a los repositorios y se definen reglas que no deben estar ni en el controlador ni en la base de datos. También gestiona excepciones y validaciones.
- Controller: Es la capa de entrada (entry point) del sistema. Define los endpoints REST que consumen los clientes web o móviles. Utiliza anotaciones como @RestController, @GetMapping, @PostMapping, etc. y responde usando objetos del tipo ResponseEntity o estructuras personalizadas como ApiResponseManager.

Esta separación modular permite que cada parte del sistema sea testeada, mantenida y extendida sin impactar a las demás. En el caso del proyecto 300 Lugares, esto es clave para desarrollar funcionalidades complejas como el filtrado por etiquetas, gestión de promociones, creación de establecimientos, autenticación JWT y control de permisos por suscripción.

ORM con Spring Data JPA y persistencia con bases de datos relacionales.

En la construcción de aplicaciones empresariales modernas, la gestión de datos persistentes es un pilar fundamental. En este sentido, los sistemas de bases de datos relacionales siguen siendo ampliamente utilizados por su consistencia, integridad y capacidad de estructurar datos de forma lógica. Para interactuar con estos sistemas desde el código Java, se utilizan tecnologías como JPA (Java Persistence API), que permite mapear objetos Java a tablas relacionales, facilitando el trabajo del desarrollador.

De acuerdo con Bauer y King (2006), la programación orientada a objetos y el modelo relacional tienen naturalezas distintas, por lo que es necesario un mecanismo que actúe como puente entre ambas. Este es precisamente el papel del mapeo objeto-relacional (ORM). Un ORM permite a los desarrolladores trabajar con bases de datos mediante clases e instancias en lugar de SQL puro, delegando al framework tareas como la generación de consultas, el control de transacciones y el mapeo de resultados.

De acuerdo con Ambler (2009), un ORM (Object-Relational Mapping o Mapeo Objeto-Relacional) es una técnica que permite interactuar con bases de datos relacionales utilizando objetos del lenguaje de programación, eliminando la necesidad de escribir directamente instrucciones SQL para operaciones básicas. Esta técnica es fundamental en el desarrollo backend moderno, ya que facilita la persistencia de datos, reduce errores y mejora la mantenibilidad del código.

Según Fowler (2003), los ORM permiten definir entidades de negocio como clases, las cuales se mapean directamente a las tablas de la base de datos, haciendo posible realizar operaciones CRUD (crear, leer, actualizar, eliminar) de forma declarativa.

En el marco de desarrollo Java, uno de los ORM más potentes y populares es Spring Data JPA, una extensión de Spring Framework que simplifica la implementación de repositorios JPA al reducir la cantidad de código necesario para acceder a los datos. Tal y como explica Baeldung et al. (2023), esta herramienta proporciona un conjunto de interfaces predefinidas que abstraen la complejidad del acceso a datos y permiten la creación automática de consultas basadas en el nombre del método. Spring Data JPA está construido sobre Hibernate, que es la implementación de referencia de JPA (Java Persistence API), esto permite mantener compatibilidad con estándares y ampliar las capacidades del sistema sin comprometer la portabilidad ni el rendimiento.

En el backend de la plataforma 300 Lugares, Spring Data JPA es utilizado en múltiples módulos:

- Realizar inserciones, búsquedas, eliminaciones y actualizaciones de datos de manera sencilla.
- Acceder a catálogos como tipos de establecimiento, suscripciones y etiquetas.
- Interactuar con tablas de uso frecuente como usuarios, empresas y promociones.
- Separar responsabilidades usando el patrón Model–Repository–Service–Controller, lo cual mejora la escalabilidad y mantenibilidad del proyecto.

Además, cuando las operaciones requieren lógica compleja o afectaban múltiples tablas (como insertar un nuevo establecimiento junto a sus etiquetas y tipos), se utilizaban procedimientos almacenados en combinación con JPA para mantener la integridad de los datos.

3. Desarrollo.

La implementación del procesamiento de imágenes fue programada utilizando Matlab R2013a versión 8.1.0.604, la adquisición, la segmentación, técnicas para tratamiento de la imagen y la visualización del proceso a través de una interfaz gráfica.

Contenerización con Docker y despliegue en la nube.

En la actualidad, el desarrollo de aplicaciones modernas exige métodos eficientes y escalables para el empaquetado y despliegue de software. Docker ha emergido como una de las herramientas más populares para lograrlo gracias a su capacidad para contenerizar aplicaciones y todas sus dependencias en un entorno aislado, portátil y reproducible.

Despliegue en Render.

Una vez construida la imagen Docker, el paso siguiente fue desplegarla en la nube. Para este propósito, se eligió Render, una plataforma de nube como servicio (PaaS) que permite implementar aplicaciones contenerizadas sin necesidad de configurar una infraestructura compleja.

Render admite despliegues desde repositorios GitHub y permite establecer flujos CI/CD (integración y entrega continua) en los cuales cada nuevo commit puede actualizar automáticamente la versión productiva. Esta estrategia de despliegue garantiza que los cambios se integren rápidamente y de forma controlada.

Según Render (2024), las aplicaciones Docker pueden configurarse con variables de entorno, volúmenes persistentes, conexiones seguras (HTTPS) y escalado automático si se desea.

JSON Web Tokens (JWT) y autenticación basada en tokens.

En el desarrollo del backend del sistema 300 Lugares, uno de los elementos fundamentales fue la implementación de un sistema de autenticación basada en tokens, específicamente con JSON Web Tokens (JWT). Esta técnica es ampliamente utilizada en sistemas distribuidos y aplicaciones modernas que requieren autenticación sin mantener sesiones en el servidor.

De acuerdo con Jones et al. (2015), el JWT es un estándar abierto (RFC 7519) que define una manera compacta y segura de transmitir información entre partes como un objeto JSON. Este token se firma digitalmente (normalmente usando HMAC-SHA o RSA) para asegurar que los datos no hayan sido alterados.

Según Auth0 (2024), un JWT consta de tres partes codificadas en Base64:

- Header (encabezado): indica el tipo de token y el algoritmo de firma.
- Payload (carga útil): contiene los "claims" o declaraciones, como el ID del usuario, su rol o su email.
- Signature (firma): asegura la integridad y autenticidad del token.
- Cuando un usuario inicia sesión, el servidor genera un JWT y se lo envía. Este token se guarda en el cliente (típicamente en localStorage o sessionStorage en aplicaciones web o en la memoria local de la app móvil) y se adjunta en cada petición HTTP en el encabezado Authorization: Bearer <token>.

En el sistema backend de 300 Lugares, se implementa Spring Security con un filtro personalizado que intercepta las solicitudes, valida el token JWT recibido y permite o deniega el acceso al recurso según los permisos codificados en

el token. Este mecanismo es especialmente importante para separar los privilegios entre clientes y administradores de la plataforma.

Cada vez que un cliente se autentica correctamente mediante su correo electrónico y contraseña, el sistema genera un JWT que incluye su ID, su nivel de suscripción y otros datos relevantes. Este token tiene una expiración definida (por seguridad) y es validado en cada llamada al API, asegurando que sólo usuarios válidos accedan a sus recursos.

Documentación.

En el desarrollo de APIs RESTful modernas, la documentación clara, estructurada y accesible es fundamental para asegurar que distintos componentes como equipo frontend, móvil, testers, mantenedores futuros puedan interactuar correctamente con el backend. En ese sentido, Swagger y su especificación OpenAPI se han convertido en estándares de facto para la definición, documentación y consumo de APIs.

Swagger, ahora reconocida formalmente como OpenAPI Specification (OAS), es una especificación independiente del lenguaje que permite describir una API REST incluyendo sus recursos, rutas (endpoints), métodos HTTP, parámetros de entrada y salida, tipos de datos, códigos de estado, métodos de autenticación, y otros detalles esenciales.

Cuando una API está descrita mediante un archivo OAS (formato JSON o YAML), esta descripción actúa como un contrato formal entre el backend y los clientes (móvil, web, otros servicios), desacoplando la implementación interna del servidor de la interfaz pública. Así, los consumidores de la API pueden conocer con precisión qué endpoints existen, qué datos esperan, qué respuestas generarán, sin necesidad de examinar el código fuente.

En el backend de 300 Lugares, documentar las APIs con Swagger permite definir claramente cada recurso expuesto de establecimientos, clientes, autenticación, filtros, promociones con sus rutas, métodos HTTP, parámetros de entrada / salida y códigos de estado. Esto facilitó enormemente la integración con el frontend web y la app móvil, pues ambos clientes consumen los mismos endpoints conforme a un contrato común.

Además, la documentación generada con Swagger sirve como referencia técnica para pruebas, mantenimiento y futuras ampliaciones del sistema, por ejemplo, cuando se incorporen nuevas funcionalidades como sistema de recompensas, filtros avanzados, u otras integraciones. De esta forma, se asegura consistencia y escalabilidad, sin comprometer la claridad o la facilidad de uso de la API.

Seguridad en API's REST y autenticación con JWT.

La seguridad es un componente esencial en cualquier sistema backend expuesto públicamente. Para el backend de 300 Lugares, es indispensable aplicar buenas prácticas de seguridad sobre sus APIs RESTful, con el fin de proteger la integridad de los datos y garantizar que solo usuarios autorizados puedan acceder a funcionalidades sensibles.

En el contexto de 300 Lugares, se hace uso de tokens JSON Web Token (JWT) para manejar la autenticación y autorización de usuarios. Este enfoque resulta conveniente en arquitecturas cliente-servidor modernas, ya que permite mantener al servidor "stateless" —es decir, sin necesidad de mantener sesiones en memoria— delegando la validación de identidad a la verificación del token en cada petición.

Según las mejores prácticas descritas por especialistas de seguridad, para implementar JWT de forma segura es indispensable:

- Utilizar claves secretas robustas al firmar los tokens;
- Verificar la firma del token en cada petición, así como su vigencia (fecha de expiración), para evitar usos indebidos o tokens robados.
- No incluir información sensible en el payload del token, ni exponer datos críticos en las URL.

Este esquema asegura que solo usuarios autenticados y autorizados puedan realizar operaciones que modifiquen datos sensibles, por ejemplo, editar establecimiento, obtener perfil de un cliente, modificar promociones y previene escenarios de manipulación o acceso no autorizado.

Otras medidas de seguridad recomendadas.

Además de la autenticación con JWT, la literatura sobre seguridad en APIs REST sugiere otras prácticas complementarias importantes:

- Aplicar limitación de tasa (rate-limiting) en los endpoints para evitar ataques de fuerza bruta o uso abusivo de la API.
- Implementar políticas de autorización por rol o permisos, de modo que no todos los usuarios autenticados tengan los mismos privilegios.
- Mantener un registro de auditoría (logs) de accesos y errores, para detectar actividades sospechosas o brechas de seguridad.

Estos mecanismos ayudan a fortalecer la seguridad del backend, reducen la superficie de ataques y establecen una base robusta para crecimiento y mantenimiento, especialmente en aplicaciones con múltiples clientes (móvil, web) como 300 Lugares.

De manera general la arquitectura del ecosistema “300 Lugares”, en el cual se integran los distintos componentes tecnológicos que conforman la solución final. En el centro se observa la plataforma backend, responsable de concentrar la lógica de negocio y de actuar como núcleo de comunicación entre la aplicación móvil desarrollada en Flutter, la aplicación web administrativa basada en Angular y la base de datos relacional MySQL. La interacción entre estos componentes se realiza mediante servicios REST seguros, empleando autenticación basada en JSON Web Tokens (JWT) para el control de acceso. Como se ilustra en la Figura 1.

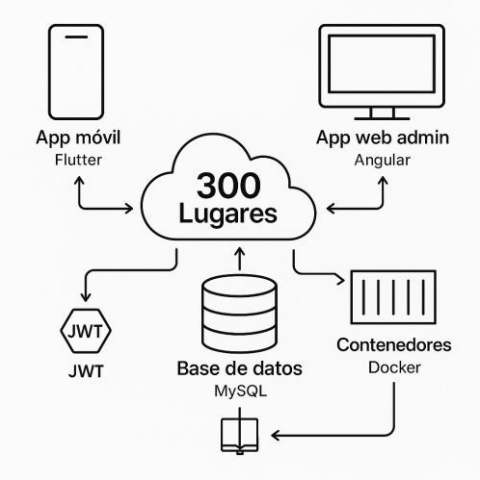

Figura 1. Ecosistema 300 lugares App.

La participación desarrollada dentro del proyecto se enfocó exclusivamente en el lado backend, abarcando el diseño e implementación de la arquitectura del servidor, la definición y construcción de la base de datos, el desarrollo de las APIs REST y la integración de los mecanismos de seguridad. Asimismo, el backend fue encapsulado en contenedores Docker, lo que permitió estandarizar el entorno de ejecución y facilitar su despliegue y mantenimiento. Esta arquitectura constituye el soporte técnico que habilita el funcionamiento integral del sistema, garantizando la interoperabilidad, seguridad y escalabilidad de las aplicaciones cliente que consumen los servicios.

Describiendo de manera integral en general del backend del sistema “300 Lugares”, se integra la arquitectura lógica y técnica implementada durante el desarrollo del proyecto. En este esquema se representa al backend como el núcleo del

ecosistema, desplegado dentro de un contenedor Docker en la nube de aplicación, donde se ejecuta una aplicación basada en Java 17 y Spring Boot. Este entorno concentra la configuración de seguridad, autenticación y documentación, a través de componentes como SecurityConfiguration, JwtAuthenticationFilter, SwaggerConfig y ApplicationConfiguration, como se muestra el diagrama en la figura 2.

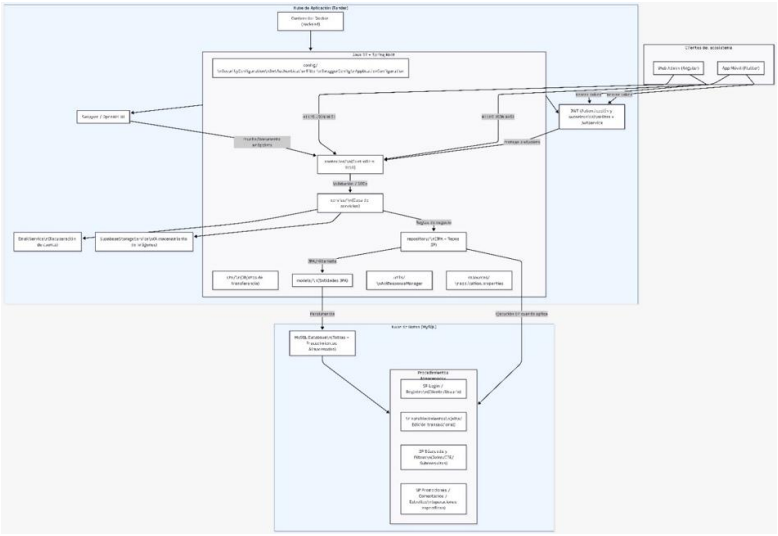

Figura 2. Diagrama de arquitectura backend.

El diagrama detalla el flujo de comunicación entre los clientes del sistema —la aplicación móvil en Flutter y la aplicación web administrativa en Angular— y la capa de controladores REST, mediante peticiones HTTPS con formato JSON. La autenticación y autorización se gestionan mediante JWT, protegiendo los endpoints y garantizando el acceso controlado a los servicios. A partir de los controladores, la información fluye hacia la capa de servicios, donde se implementan las reglas de negocio, validaciones y orquestación de procesos, apoyándose en objetos DTO para la transferencia segura de datos.

Asimismo, el diagrama muestra la interacción con la capa de persistencia, implementada mediante Spring Data JPA e Hibernate, así como el uso de procedimientos almacenados en la base de datos MySQL para operaciones críticas, consultas complejas y transacciones. Se incluyen también servicios externos de apoyo, como el envío de correos electrónicos para recuperación de cuentas y el almacenamiento de imágenes en Supabase. En conjunto, este diagrama contextualiza la arquitectura backend desarrollada, evidenciando una estructura modular, segura y escalable, alineada con las responsabilidades asumidas en el proyecto y con los objetivos técnicos definidos para el sistema.

Como resultado del desarrollo, donde el backend del ecosistema “300 Lugares” opera bajo un esquema de inyección de dependencias administrado por el contenedor IoC de Spring (ApplicationContext). En términos conceptuales, el contenedor actúa como el componente central de ensamblaje: identifica clases marcadas con estereotipos de Spring (por ejemplo, @RestController, @Service, @Repository y @Configuration), crea instancias (beans) de manera controlada y resuelve automáticamente sus dependencias, con lo cual se evita el acoplamiento directo entre módulos y se garantiza un flujo estructurado de interacción. Como se aprecia el diagrama de la figura 3.

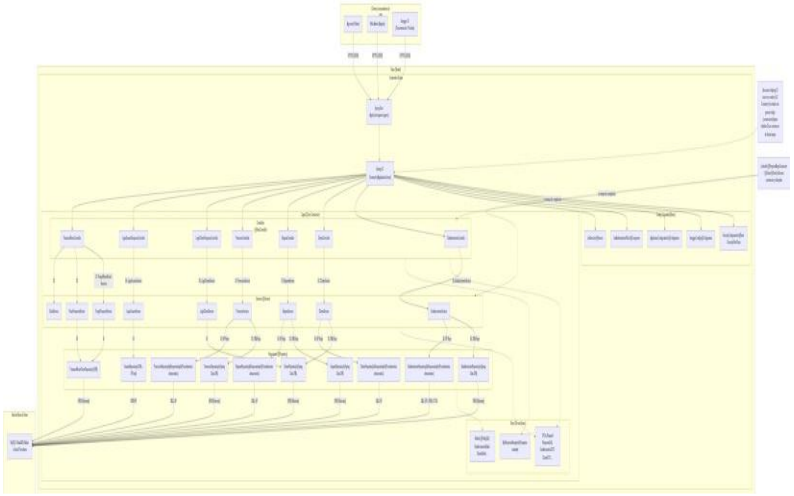

Figura 3. Patrón de inyección de dependencias.

A nivel funcional, el resultado observable es una separación clara de responsabilidades. La capa de Controllers concentra la exposición de endpoints y la recepción de solicitudes HTTP/JSON. La capa de Services encapsula la lógica de negocio y orquesta los casos de uso. Finalmente, la capa de Repositories materializa el acceso a datos, combinando dos estrategias: (1) persistencia estándar mediante Spring Data JPA/Hibernate para operaciones directas y (2) repositorios especializados que ejecutan procedimientos almacenados para consultas complejas o transacciones que involucran múltiples tablas. Esta segmentación permite que cada capa evolucione con impacto controlado sobre las demás, manteniendo un modelo de mantenimiento y ampliación más predecible.

Adicionalmente, el diagrama refleja la presencia de un cinturón transversal de configuración y seguridad, donde Spring Security intercepta solicitudes mediante un filtro JWT antes de que alcancen los controladores. Este comportamiento confirma que el control de acceso se integra como una capacidad del backend y no como lógica dispersa en cada endpoint, fortaleciendo consistencia y gobernanza del acceso.

Finalmente, se documenta el rol de Lombok como un apoyo de productividad: su contribución es generar automáticamente constructores y código repetitivo en compilación, facilitando la adopción sistemática de inyección por constructor. Es importante señalar que Lombok no realiza inyección; el mecanismo de inyección y el ciclo de vida de objetos corresponden al contenedor IoC, mientras que Lombok únicamente reduce el “boilerplate” necesario para mantener el patrón aplicado de forma uniforme.

El diagrama entidad–relación, describe la estructura lógica de la base de datos diseñada para el sistema “300 Lugares”, la cual constituye el soporte de persistencia de toda la información gestionada por el backend. En este esquema se representa a p_establecimiento como la entidad central del modelo, ya que alrededor de ella se articulan la mayoría de las relaciones que permiten describir de forma integral a cada negocio registrado dentro de la plataforma. como se presenta en la figura 4.

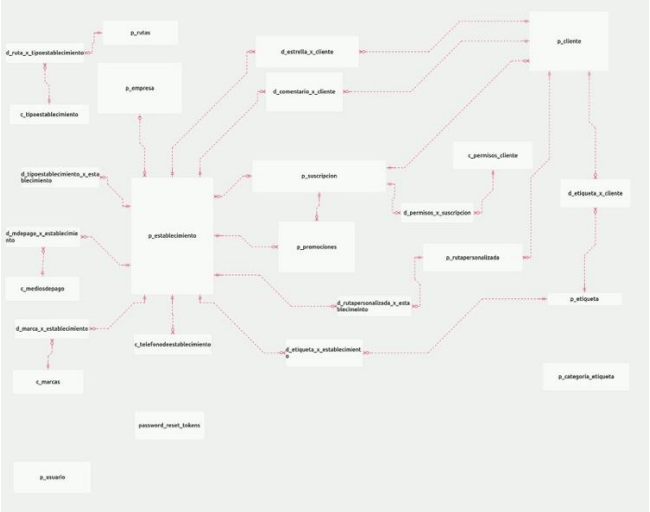

Figura 4. Diagrama entidad relación de la base de datos final.

El diagrama muestra cómo los establecimientos se vinculan con empresas, tipos de establecimiento, marcas, formas de pago, teléfonos y etiquetas, utilizando tablas intermedias que resuelven relaciones de tipo muchos a muchos, como d_tipoestablecimiento_x_establecimiento, d_marca_x_establecimiento y d_etiqueta_x_establecimiento. Esta normalización permite flexibilidad en la clasificación de los establecimientos y facilita la implementación de filtros avanzados dentro de la aplicación móvil y del sistema administrativo.

Asimismo, el modelo integra la gestión de clientes, quienes pueden interactuar con los establecimientos mediante calificaciones y comentarios, representados por las entidades d_estrella_x_cliente y d_comentario_x_cliente. El sistema de suscripciones y permisos se encuentra modelado a través de las entidades p_suscripcion, c_permisos_cliente y d_permisos_x_suscripcion, lo que permite controlar de manera estructurada el acceso a funcionalidades dentro de la plataforma, en función del tipo de suscripción asignada a cada cliente.

El diagrama también contempla la gestión de promociones, rutas personalizadas, etiquetas categorizadas y preferencias del cliente, así como componentes de seguridad como password_reset_tokens y la entidad p_usuario para el control administrativo. En conjunto, este modelo entidad–relación refleja un diseño orientado a la escalabilidad y consistencia de los datos, alineado con los requerimientos funcionales del backend y con el uso de procedimientos almacenados y JPA para garantizar la integridad y eficiencia en las operaciones de persistencia.

El flujo completo de autenticación implementado en el backend del sistema 300 Lugares, el cual fue desarrollado utilizando Java 17 y Spring Boot, siguiendo los principios de seguridad y desacoplamiento propios de una arquitectura moderna basada en servicios REST, como se logra apreciar en el diagrama de secuencia que se visualiza en la figura 5.

El proceso inicia cuando el usuario, desde la aplicación móvil o desde la aplicación web administrativa, envía sus credenciales de acceso (correo electrónico o número telefónico y contraseña) al backend a través de un endpoint de autenticación expuesto por los controladores REST. Estas solicitudes son recibidas y validadas inicialmente en la capa de controladores, verificando el formato y la presencia de los datos requeridos.

Posteriormente, la solicitud es delegada a la capa de servicios de autenticación, donde se concentra la lógica de negocio relacionada con el inicio de sesión. En esta etapa, el sistema consulta la información del usuario o cliente en la base de datos, ya sea mediante repositorios JPA o, en los casos correspondientes, mediante procedimientos almacenados definidos en MySQL. La verificación de credenciales se realiza comparando el hash de la contraseña almacenada con el valor proporcionado por el usuario.

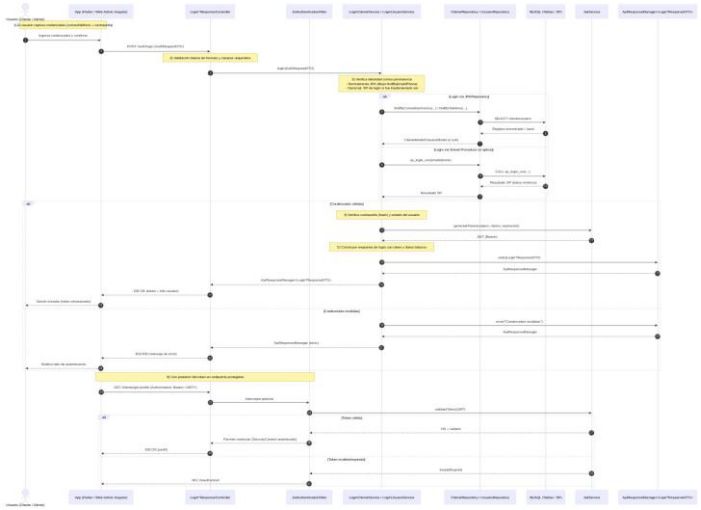

Figura 5. Flujo completo de login con JWT.

Una vez validadas las credenciales, el backend genera un JSON Web Token (JWT) mediante el servicio especializado de generación de tokens. Este token encapsula la identidad del usuario, su rol y los metadatos necesarios para controlar el acceso a los recursos del sistema, estableciendo así un mecanismo de autenticación sin estado (stateless). El token generado es devuelto al cliente dentro de una respuesta estructurada, permitiendo que sea utilizado en las solicitudes subsecuentes.

En las operaciones posteriores, cada petición protegida incluye el token JWT en el encabezado de autorización. Dichas peticiones son interceptadas por un filtro de seguridad, el cual valida la integridad y vigencia del token antes de permitir el acceso a los endpoints correspondientes. En caso de que el token sea válido, el contexto de seguridad es establecido y la solicitud continúa su flujo normal; de lo contrario, el acceso es denegado.

Este esquema de autenticación permitió implementar un control de acceso robusto, escalable y desacoplado, garantizando la protección de los servicios del backend y asegurando que tanto la aplicación móvil como el sistema web administrativo operaran bajo un mismo modelo de seguridad centralizado. La implementación de JWT representó un componente clave dentro de los resultados del proyecto, al proporcionar una base sólida para la gestión segura de usuarios y sesiones dentro del ecosistema 300 Lugares.

Conclusiones.

El desarrollo del backend para el ecosistema de 300 Lugares representó un reto técnico y organizativo que implicó el diseño y construcción de una arquitectura sólida, escalable y alineada con las necesidades reales del proyecto. A lo largo del proceso se implementaron buenas prácticas de desarrollo, como el uso de patrones de diseño (Model-Service-Repository), inyección de dependencias, arquitectura RESTful, seguridad mediante JWT y documentación estructurada con Swagger.

Uno de los logros más importantes fue la creación de una base de datos robusta en MySQL con procedimientos almacenados diseñados para garantizar integridad de datos, eficiencia y control transaccional. Además, el uso de Docker permitió la contenerización del backend para su despliegue y pruebas, asegurando portabilidad y facilidad de administración.

El trabajo desarrollado contribuyó de manera sustancial a la funcionalidad del proyecto completo, permitiendo que la app móvil y la plataforma web se conectaran eficazmente a un backend bien estructurado. Esta integración hizo

posibles operaciones como el registro de clientes, autenticación segura, consultas avanzadas mediante filtros, administración de establecimientos y gestión de promociones.

Recomendaciones:

- Para futuras iteraciones del proyecto, se recomienda implementar pruebas automatizadas (unitarias y de integración) para garantizar mayor estabilidad en cada despliegue.
- Es conveniente migrar gradualmente los servicios hacia una arquitectura basada en microservicios, conforme el número de usuarios crezca.
- Integrar herramientas de monitoreo y análisis de logs (como ELK Stack o Prometheus) para facilitar el mantenimiento y la detección temprana de errores.
- Desarrollar un entorno CI/CD que automatice los procesos de compilación, pruebas y despliegue para garantizar entregas continuas de calidad.
- Documentar formalmente todos los endpoints y procedimientos almacenados en un repositorio compartido para facilitar el onboarding de nuevos desarrolladores.

Créditos.

Los autores agradecen a la empresa Ambar Rojo Studios, S.A de C.V y al Tecnológico Nacional de México por las facilidades para el desarrollo del proyecto.

Referencias bibliográficas.

Ambler (2006) Scott Ambler. *The Object-Relational Impedance Mismatch. 2006*. http://www.agiledata.org/essays/impedanceMismatch.html

Alur, D., Crupi, J., & Malks, D. (2003). *Core J2EE Patterns: Best Practices and Design Strategies* (2nd ed.). Prentice Hall.

Auth0. (2024). *JWT Introduction***.** https://auth0.com/docs/secure/tokens/json-web-tokens

Baeldung. (2023). *Guide to Spring Data JPA.* https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

Bauer, C., & King, G. (2006). *Java Persistence with Hibernate. Manning Publications***.**

CANIRAC (2023). *Conociendo a la industria restaurantera.* https://portal.canirac.org.mx/noticias/conociendo-a-la-industria-restaurantera/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

INEGI (2023). https://www.inegi.org.mx/app/biblioteca/ficha.html?upc=702825198138

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force*. https://datatracker.ietf.org/doc/html/rfc7519

Render. (2024). *Deploy Docker Services.* https://render.com/docs/deploy-docker-services

Secretaría de Turismo del Estado de Chiapas (2022) https://turismochiapas.gob.mx/storage/paginas/cuaderno-estadistico/secciones/informacion-estadistica-2018/1719979869_mayo-2018.pdf

Spring. (2024). *Dependency Injection in Spring Framework*. https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

Spring. (2024). *JWT Authentication with Spring Security.* https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html

Información de los autores.

Pablo Jesús Gómez Pérez . Alumno de la carrera de Ingeniería en Sistemas Computacionales con experiencia en desarrollo backend, soporte aplicativo, atención de incidencias, mantenimiento evolutivo y correctivo, y aseguramiento operativo de sistemas empresariales. He trabajado con Java, Spring Boot, Spring Data JPA, Maven, Swagger, MySQL, SQL Server, C# y tecnologías web para desarrollar y dar soporte a APIs, plataformas administrativas y aplicaciones usadas en operación real. Es fuerte en análisis técnico, diagnóstico de fallas, debugging, validación de soluciones, liberación de cambios y continuidad operativa. Cuento además con formación contable, lo que me permite comprender mejor procesos financieros y entornos de alta responsabilidad operativa.

María Guadalupe Monjarás Velasco. María Guadalupe Monjarás Velasco obtuvo el grado de doctora en sistemas computacionales en el año 2012, tiene una maestría en ciencias de la computación obteniendo el título en el año 2009, es ingeniera en sistemas computacionales egresada y titulada en el año 2006, es Profesora e Investigadora en el área de Ingeniería en Sistemas Computacionales. Actualmente, se desempeña como Líder del Cuerpo Académico "Innovación en Tecnologías de Información y Comunicación Aplicadas" y cuenta con el reconocimiento de Perfil Deseable PRODEP. Su experiencia en gestión académica incluye haber servido como Jefa del Departamento de Sistemas y Computación en el Instituto Tecnológico de Tuxtla Gutiérrez (período 2016-2025), así como integrante de su Comité Académico durante el mismo lapso. Sus líneas de investigación actuales se centran en la Inteligencia Artificial, el Procesamiento de Lenguaje Natural (NLP) y la implementación de arquitecturas RAG para entornos educativos y administrativos.

Imelda Valles López. Desde 1991, profesora del área de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tuxtla Gutiérrez. Maestro en Administración por el I.T.T.G., en 2001 e Ingeniero en Sistemas Electrónicos por el ITESM en 1990.Docente de las materias de Redes, Teoría Matemática de la Computación y Compiladores. Integrante del cuerpo académico "Innovación en Tecnologías de Información y Comunicación Aplicadas".

Elfer Isaías Clemente Camacho es Ingeniero en sistemas computacionales, por el Tecnológico de Tuxtla Gutiérrez, cuanta con Maestría en desarrollo de software por el Instituto de Estudios Superiores de Chiapas. Doctor en Tecnología Educativa por la Universidad del País INNOVA. Docente desde 2019, en la Carrera de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tuxtla Gutiérrez. Docente en la Universidad Autónoma de Chiapas desde 2013. Docente en la Universidad Pablo Guardado Chávez desde 2020. Profesor con Perfil Deseable (PRODEP) desde 2025.

Ciclalli Cabrera García Es doctora en ingeniería aplicada obteniendo el grado en el 2020, el grado de Maestra en tecnologías en informática en el 2012, termino la carrera de Licenciatura en informática en el 2008. Actualmente asesora proyectos académicos y en concursos académicos, miembro del Cuerpo Académico "Innovación en Tecnologías de Información y Comunicación Aplicadas"

Raúl Parades Trinidad es catedrático y empresario, es Ingeniero en sistemas computacionales por el Tecnológico Nacional de México Campus Tuxtla Gutiérrez, egresado en el año de 1995. Obtuvo el grado de Maestro en Tecnología Educativa por la Universidad Virtual del Instituto Tecnológico y de Estudios Superiores de Monterrey, en el año 2006. Cuenta con 27 años de experiencia en el sector académico, impartiendo clases en el área de Ingeniería en Sistemas Computacionales del Tecnm Campus Tuxtla Gutiérrez. Así mismo ha participado en actividades de investigación académica, implementando 2 proyectos de investigación en calidad de director. En el año 2011 creó 2 empresas de TI, especializadas en el desarrollo de software a la medida y en medios interactivos: Bitzero Technologies SA de CV y Ambar Rojo Studios SA de CV. En marzo de 2012 Fundamentos el Clúster de TI Chiapas. Actualmente preside la Comisión de Innovación e IA de Coparmex Chiapas.