I.E.S GRAN CAPITAN

GRADO FORMATIVO DE GRADO SUPERIOR
EN APLICACIONES MULTIPLATAFORMA.

PROYECTO INTEGRADO

Manual Técnico

Autores:

- Antonio Jesús Martínez Díaz – 2º DAM
- Rocío Zuara Jiménez – 2º DAM
- Lorena Barea Rot – 2º DAM

INDICE:

1.INTRODUCCION:

Lavadero Sepúlveda es un sistema integral y multiplataforma diseñado para la gestión completa de citas en lavaderos de vehículos. El proyecto combina aplicaciones de escritorio, web y móvil, permitiendo digitalizar por completo los procesos de reserva, administración y atención al cliente. Su enfoque modular garantiza una experiencia fluida tanto para los usuarios como para el personal del negocio, adaptándose a las necesidades actuales de eficiencia, accesibilidad y modernización.

2. OBJETIVOS

Objetivos Generales

Desarrollar una plataforma flexible, moderna y centralizada que permita gestionar de forma eficiente las citas, clientes, servicios y recursos de un lavadero de vehículos, optimizando el flujo de trabajo, reduciendo tareas manuales y ofreciendo una experiencia de uso ágil e intuitiva.

Objetivos Específicos

1. Digitalización completa del proceso de reservas

El sistema permitirá a los clientes reservar, modificar o cancelar citas de manera online desde cualquier dispositivo. Esto eliminará la gestión manual, reducirá errores, agilizará los tiempos de atención y mejorará la organización diaria del personal.

Incluye funcionalidades como:

- Control de disponibilidad en tiempo real.
- Registro automatizado de citas.
- Gestión de servicios y horarios.

2. Desarrollo de un sistema multiplataforma

La plataforma ofrecerá varios puntos de acceso para garantizar la máxima accesibilidad:
Aplicación web para clientes y administradores.
Aplicación móvil (Android e iOS) orientada a clientes y personal.
Aplicación de escritorio para la gestión operativa interna.
Todas las interfaces estarán sincronizadas mediante una arquitectura centralizada.

3. Centralización y sincronización de la información

Toda la información del negocio —clientes, empleados, citas, servicios, historial y métricas— se almacenará en una base de datos centralizada, asegurando coherencia y disponibilidad en tiempo real.

Esto permitirá:

- Acceso unificado a los datos.
- Informes automáticos, estadísticas y seguimiento operativo.
- Evitar duplicidades o inconsistencias.

4. Mejora de la eficiencia operativa y de la experiencia del cliente

La plataforma incorporará herramientas que agilizan la interacción con el cliente y la organización interna:

- Notificaciones automáticas por correo o SMS.
- Recordatorios de citas y avisos de cambios.
- Gestión optimizada de turnos y cargas de trabajo.
- Reducción de tiempos muertos y colas.

5. Flexibilidad, modularidad y escalabilidad

El sistema se diseñará para permitir la incorporación de nuevas funcionalidades sin afectar su funcionamiento.

Entre las posibles extensiones futuras se incluyen:

- Integración con pasarelas de pago.
- Control de inventario.
- Programas de fidelización.
- Expansión a franquicias o múltiples sedes.

6. Seguridad y protección de datos

Se implementarán medidas de seguridad que garanticen la protección de la información y el cumplimiento del RGPD:

- Autenticación segura.
- Cifrado de contraseñas.
- Conexiones HTTPS.
- Gestión segura de roles y permisos.

7. Durabilidad, mantenibilidad y buenas prácticas

El proyecto se desarrollará con tecnologías estándar y ampliamente soportadas, asegurando:

- Código limpio y documentado.
- Arquitectura modular para facilitar actualizaciones y correcciones.
- Mantenimiento eficiente sin interrumpir la actividad diaria del negocio.

3. ESTUDIO PREVIO

3.1. Estado actual

En la actualidad, la mayoría de los lavaderos de vehículos continúan gestionando sus citas de forma manual, a través de agendas físicas, notas improvisadas o mediante atención telefónica directa. Este modelo, aunque tradicional, presenta importantes limitaciones que afectan tanto al funcionamiento interno como a la calidad del servicio ofrecido al cliente.

Entre los principales problemas detectados se encuentran:

- **Confusión de horarios y solapamiento de reservas:**

La ausencia de un sistema automatizado provoca que, en momentos de alta demanda, las citas puedan anotarse de forma incorrecta o duplicada. Esto genera descoordinación, retrasos y una percepción negativa por parte del cliente.

- **Tiempos elevados en la atención y registro de citas:**

El personal debe invertir tiempo en atender llamadas, confirmar disponibilidad, buscar huecos libres y registrar manualmente cada cita. Este proceso no solo es lento, sino que aumenta el riesgo de errores humanos.

- **Falta de control y seguimiento del historial de clientes y servicios realizados:**

Los sistemas manuales dificultan llevar un registro detallado de servicios previos, preferencias del cliente, frecuencia de visitas o incidencias. Sin esta información, es más complejo ofrecer un trato personalizado o realizar acciones comerciales efectivas.

- **Dificultad para obtener estadísticas precisas de ocupación y demanda:**

Sin una base de datos centralizada y sin herramientas de análisis, resulta prácticamente imposible medir la carga de trabajo, la estacionalidad, los servicios más solicitados o las horas punta. Esto limita la toma de decisiones estratégicas y la planificación del negocio.

Estas limitaciones afectan directamente a la **eficiencia operativa**, provocando desorganización, falta de visibilidad del flujo de trabajo y pérdidas de tiempo. Al mismo tiempo, repercuten en la **experiencia del cliente**, que puede percibir desatención, tiempos de espera prolongados o falta de profesionalidad.

Por todo ello, surge la necesidad de **digitalizar, automatizar y centralizar la gestión de citas**, con el fin de optimizar el funcionamiento interno del lavadero y ofrecer un servicio más rápido, transparente y profesional.

3.2. Estudio de soluciones existentes

Para definir la solución tecnológica más adecuada, se realizó un análisis de diversas herramientas y plataformas de gestión de reservas actualmente disponibles en el mercado. El objetivo fue evaluar si alguna de ellas podía adaptarse directamente a las necesidades de los lavaderos, tomar como referencias y saber que mejorar en nuestro propio proyecto.

Los criterios analizados fueron:

- **Soluciones de código abierto vs. software propietario:**

Se valoró la libertad de personalización, los costos asociados y la capacidad de modificar o ampliar funcionalidades según las necesidades del negocio.

- **Facilidad de uso y accesibilidad multiplataforma:**

Muchas herramientas ofrecen aplicaciones web o móviles, pero no siempre permiten una integración fluida con sistemas de escritorio o con procesos internos específicos.

- **Flexibilidad para adaptarse a los procedimientos concretos del lavadero:**

Algunas plataformas están pensadas para peluquerías, talleres o servicios generales, y no contemplan particularidades propias de un lavadero, como tiempos variables según el tipo de servicio, vehículos especiales o gestión dinámica de disponibilidad.

- **Capacidad de integración con otros sistemas:**

Se valoró la posibilidad de conectar las plataformas con sistemas de pago, CRM, inventarios o herramientas internas del negocio.

Tabla comparativa de soluciones analizadas

APP	Características principales	Ventajas	Limitaciones
Picktime	• Reservas online 24/7 • Gestión de personal • Opciones de pago integradas	• Accesible a cualquier hora • Configuración sencilla • Adecuado para negocios pequeños y medianos	• Personalización limitada • Algunas funciones avanzadas requieren pago • Adaptación genérica, no específica para lavaderos
Booker	• Disponibilidad en tiempo real • Recordatorios automáticos • Pagos integrados	• Plataforma robusta y estable • Automatización de avisos • Integración con múltiples servicios	• Coste mensual elevado según volumen • Curva de aprendizaje moderada • Poca flexibilidad para flujos de trabajo específicos
EasyWeek	• Calendario accesible desde cualquier dispositivo • Gestión de horarios • Sistema multiusuario	• Muy accesible y sencillo de usar • Interfaz moderna • Adecuado para equipos distribuidos	• Personalización limitada • Funciones avanzadas en planes superiores • No pensado para servicios con tiempos variables como lavaderos
Vev	• Pensado para servicios móviles • Planificación inteligente • Recordatorios automáticos	• Ideal para negocios dinámicos • Buena automatización y avisos • Diseño minimalista	• Enfoque no específico para lavaderos • Escalabilidad limitada • Integración reducida con sistemas externos
Urable / Zenbooker / MioCommerce	• CRM integrado • Gestión de clientes, citas y pagos • Funciones de marketing	• Más completas a nivel comercial • Seguimiento detallado del cliente • Incluyen herramientas de fidelización	• Coste elevado en la mayoría • Configuración más compleja • Requieren adaptación manual para procesos específicos

4. DISEÑO BACKEND

4.1. Diseño General: Arquitectura y Capas

El proyecto consiste en el desarrollo de un Sistema Integral de Gestión de Citas (SGC) para un lavadero de vehículos, implementado bajo una Arquitectura de Servicios Distribuidos con un backend centralizado.

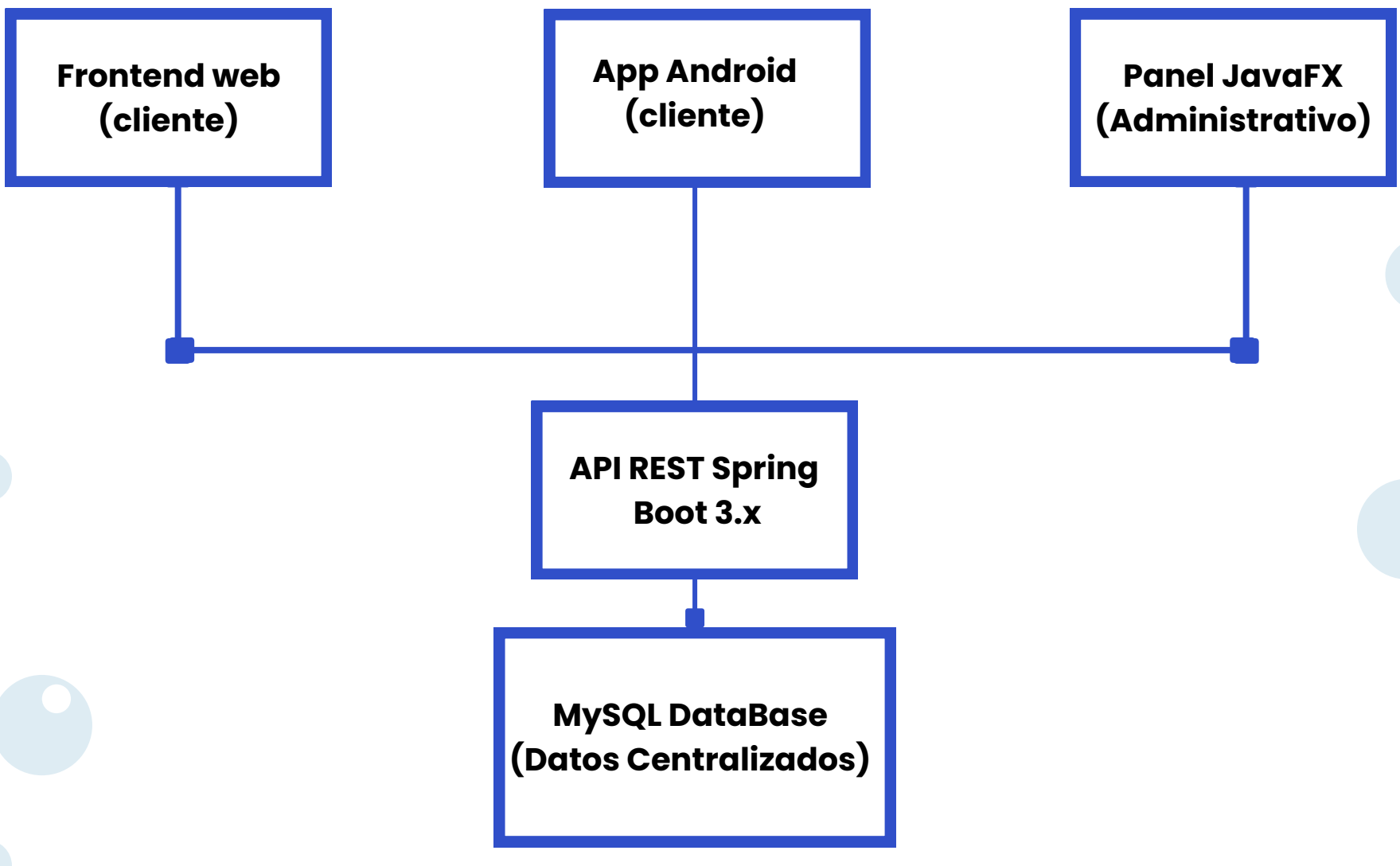
La solución adopta un API-First Design, donde la API RESTful de Spring Boot actúa como la única Capa de Lógica de Negocio (BLL) y el punto de acceso para todas las interfaces de usuario.

Modelo Arquitectónico de Cuatro Capas

El sistema se estructura en un modelo de cuatro capas que garantiza la escalabilidad, la mantenibilidad y la coherencia de los datos en tiempo real:

- **Capa de Presentación Policanal (Frontends):** Múltiples interfaces adaptadas al entorno (Web, Móvil, Escritorio) para cubrir las necesidades de clientes y administradores.
- **Capa de Servicios de Aplicación (API REST)**: Responsable de la validación, la lógica transaccional, la Clasificación Inteligente de Vehículos y la integración con servicios externos (SMTP para emails, Google Services).
- **Capa de Persistencia (BBDD):** MySQL 8.x como fuente única de verdad (Single Source of Truth) para el almacenamiento de datos relacionales (citas, tarifas, clientes, logs).
- **Capa de Infraestructura (DevOps)**: Uso de Docker y Docker Compose para la containerización, asegurando la portabilidad y el despliegue coherente del stack completo.

Diagrama Conceptual



4.2 Diseño Detallado: Implementación Técnica

4.2.1. Arquitectura de Componentes y Flujo de Datos

Se define el rol específico de cada componente, las tecnologías utilizadas y su propósito dentro del ciclo de vida de la aplicación.

Componente	Rol en el Sistema	Tecnologías Clave	Protocolo de Comunicación	Propósito y Contribución
Frontend Web	Capa de Presentación Pública.	Thymeleaf, HTML5, CSS Grid/Flexbox.	HTTP(S) hacia API REST.	Provee el **Portal de Reservas** público. Diseño **Mobile-First** y *responsive* con Thymeleaf y CSS Grid.
App Android	Capa de Presentación Móvil Nativa.	Java Nativo, Retrofit.	HTTP(S) hacia API REST.	Ofrece una **Experiencia de Usuario (UX)** optimizada. Retrofit gestiona las peticiones HTTP con alto
Panel JavaFX	Capa de Presentación Administrativa.	JavaFX, Scene Builder.	HTTP(S) hacia API REST.	Herramienta de escritorio para la **gestión avanzada** (CRUD de citas, tarifas y horarios).
API RESTful	**Backbone: Lógica de Negocio Centralizada.**	Spring Boot 3.x (MVC, Data JPA), Java Mail.	JDBC hacia BBDD.	Centraliza la **Inteligencia de Negocio** (Clasificación Vehicular, Scheduler) y es el único punto de contacto con la BBDD.
MySQL Database	Capa de Persistencia (*Single Source of Truth*).	MySQL 8.x.	JDBC (JPA).	Almacena de forma **relacional** y persistente toda la información crítica del sistema.
Docker / Compose	Capa de Infraestructura (**DevOps**).	Docker, Docker Compose.	N/A	Facilita la **containerización** y el despliegue del *stack* completo en un único comando, asegurando la **consistencia** del entorno.

4.2.2. Diseño de Lógica Central (Funcionalidades Clave)

A. Clasificación Automática de Vehículos 🚗 Implementación: Módulo dentro de la API REST (Service Layer).

- Objetivo: Determinar automáticamente la tarifa y el tiempo de servicio requerido al recibir el tipo de vehículo del cliente.
- Mecanismo: El frontend envía la información y la API aplica la lógica condicional (ej: Turismo) para devolver la categoría de servicio final.

B. Sistema de Notificaciones Asíncronas 📧

- Tecnología: Java Mail orquestado por la API REST.
- Funcionalidades: Envío de confirmaciones transaccionales inmediatas y activación de un Scheduler de Spring para el envío programado de recordatorios de cita (ej. 24h antes), minimizando ausencias (no-shows).

C. Gestión Flexible de Horarios y Disponibilidad

- Lógica: La API valida las solicitudes de reserva consultando en tiempo real la MySQL Database.
- Propósito: Garantizar que el slot solicitado por el cliente esté libre, considerando la duración variable del servicio según la clasificación del vehículo.

4.2.3. Estructura de la API REST (Controladores)

A continuación se presenta el análisis de los controladores utilizados en el proyecto:

1) CitaApiController.java:

El **CitaApiController** actúa como punto de entrada REST para la gestión de citas dentro de la aplicación. Implementa la lógica de comunicación entre el cliente (frontend o API externa) y la capa de servicio (CitaService), encargada de la persistencia de datos. Permite realizar operaciones CRUD (crear, leer, actualizar y eliminar) sobre las citas.

Método	Endpoint	Descripción
GET	/api/citas	Lista todas las citas almacenadas.
GET	/api/citas/{id}	Recupera una cita específica por su ID.
POST	/api/citas	Crea una nueva cita a partir de un objeto CitaDTO. Realiza validaciones y envía un correo de confirmación.
GET	/api/citas/horarios-disponibles?fecha=dd/MM/yyyy	Devuelve los horarios disponibles para una fecha concreta, excluyendo las 14:00.
DELETE	/api/citas/{id}	Elimina una cita existente.
GET	/api/citas/verificar-disponibilidad?fecha=...&hora=...	Comprueba si existe una cita registrada en esa fecha y hora.
GET	/api/citas/por-fecha	Agrupa todas las citas por fecha y las devuelve ordenadas cronológicamente.

2. CitaController.java

Gestiona la lógica de negocio para las reservas de citas desde la interfaz web y la interacción con el modelo Cita. Este controlador maneja la visualización de citas y la creación desde el frontend.

Método	Endpoint	Descripción	Vista
GET	/	Muestra la página principal	index
GET	/nueva-cita	Muestra formulario de crear cita, envía tipos de lavado	formulario
POST	/guardar-cita	Valida y guarda cita, calcula depósito, envía email, redirige	Redirige a /instrucciones-pago
GET	/confirmacion	Página final tras pagar	confirmacion
GET (AJAX)	/horarios-disponibles	Devuelve horas libres para una fecha	JSON (array de horas)
GET	/admin/listado-citas	Muestra citas futuras, pasadas y clientes con faltas	admin/listado-citas
GET	/admin/eliminar-cita/{id}	Elimina cita por ID	Redirige a listado
GET	/admin/marcar-asistencia/{id}/{asistio}	Registra asistencia o falta, envía email, actualiza cita	Redirige a listado
GET	/instrucciones-pago	Muestra instrucciones de cómo pagar el depósito	instrucciones-pago
POST	/confirmar-pago	Cliente confirma pago Bizum, marca depósito pagado, envía email	Redirige a confirmacion
GET	/admin/confirmar-deposito/{id}	Admin marca depósito como pagado	Redirige a listado

3. ConfiguracionController.java

Se encarga de gestionar las configuraciones del sistema, como los horarios de apertura, pudiendo ser modificados

Método	Endpoint	Descripción
GET	/api/configuracion	JSON con horarios que devuelve configuración fija de horarios del negocio.

4. GaleriaController.java

Controlador que maneja las imágenes y fotos de la galería del lavadero, permitiendo la visualización de las imágenes relacionadas con los servicios.

Método	Endpoint	Descripción
GET	/Galeria	Mostrar las fotos de los servicios y trabajos realizados.

5. HorarioController.java

Maneja la gestión de los horarios de trabajo del lavadero, permitiendo la visualización y la modificación de los horarios disponibles para las citas.

Método	Endpoint	Descripción
GET	/horario	Devuelve la vista horario.html que muestra los horarios del negocio.

6.ProductosController.java

Controlador que gestiona los productos o servicios ofrecidos por el lavadero, incluyendo la visualización, creación, edición y eliminación de productos.

Método	Endpoint	Descripción
GET	/productos	Devuelve la vista productos.html que muestra los productos del negocio.

7.ReminderController.java

Este controlador se encarga de la gestión de los recordatorios de citas para los clientes, asegurándose de que los usuarios reciban notificaciones por correo o mensajes cuando se acerque su cita.

Método	Endpoint	Descripción
GET	/admin/recordatorios/enviar/{citaId}	Envía un recordatorio manual para la cita con ID especificado.
GET	/admin/recordatorios/proceso-diario	Ejecuta manualmente el envío de recordatorios para todas las citas de mañana.

8.TarifasController.java

Gestiona las tarifas y precios de los servicios de lavado, permitiendo que los administradores modifiquen las tarifas según sea necesario.

Método	Endpoint	Descripción
GET	/tarifas	Devuelve la vista tarifas.html que muestra los tipos de lavado y sus precios.

9.Otros:

- **GoogleVerificationController.java**: Este controlador gestiona la verificación de Google para los servicios que requieren autenticación o validación externa, como la verificación de la propiedad del dominio.
- **PolicyController.java:** Se encarga de la gestión de las políticas de servicio (términos y condiciones, políticas de privacidad, etc.), que pueden ser consultadas y modificadas por los administradores.

4.2.4. Modelo de Datos Clave (Entidades JPA)

1.TipoLavado.java

Modelo que define los diferentes tipos de servicios de lavado disponibles (lavado completo, interior, exterior, etc.). Se compone de un ENUM Java que contiene todos los tipos de lavado del negocio asociándoles un precio (BigDecimal) y un Converter JPA para guardar estos valores en la base de datos.

```
public enum TipoLavado {
LAVADO_COMPLETO_TURISMO("Lavado Completo Turismo", new BigDecimal("23.0")),
....
}
```

Pero este enum no solo es un texto, también tiene datos asociados y un converter JPA para pasar de enum a texto, para la base de datos.

2.TipoLavadoConverter.java

Conversor que transforma los valores de los tipos de lavado en un formato adecuado para la base de datos o la interfaz de usuario; almacenando los valores del enum de (TipoLavado) como texto en la base de datos y recuperarlos correctamente, incluso si existen diferencias de mayúsculas, minúsculas o guiones bajos, transformándolos al formato de la BD.

Ej: `return tipoLavado.name().toLowerCase();`

3.Cita.java

Modelo que representa una cita en el sistema. Contiene detalles como el cliente, el tipo de servicio, la fecha y la hora de la cita enlazado directamente mediante JPA/Hibernate con la BD.

Campo BD	Tipo BD	Tipo Java	Descripción
id	BIGINT	Long	Identificador único de la cita.
nombre	VARCHAR	String	Nombre del cliente.
email	VARCHAR	String	Email del cliente.
telefono	VARCHAR	String	Teléfono del cliente.
modelo_vehiculo	VARCHAR	String	Modelo del vehículo.
tipo_lavado	VARCHAR	TipoLavado	Tipo de lavado seleccionado.
fecha	DATE	LocalDate	Día de la cita.
hora	TIME	LocalTime	Hora de la cita.
asistida	BOOLEAN	Boolean	Si asistió o no (true/false/null).
faltas	INT	Integer	Número de faltas acumuladas.
precio_total	DECIMAL(10, 2)	BigDecimal	Precio total del servicio.
deposito	DECIMAL(10, 2)	BigDecimal	Depósito previo (50% del total).
deposito_pagado	BOOLEAN	Boolean	Indica si el depósito fue pagado.
referencia_bizum	VARCHAR(50)	String	Código de referencia del Bizum.
fecha_pago_deposito	DATE	LocalDate	Fecha de pago del depósito.

5. Diseño Frontend

A continuación se presenta el análisis de la interfaz de usuario tanto en la aplicación de escritorio como en la de movil.

5.1 Interfaz de Escritorio:

El frontend de esta aplicación web está construido usando:

- HTML5 → para la estructura de las páginas
- CSS3 → para el estilo y apariencia visual
- JavaScript → para funcionalidades dinámicas
- Thymeleaf → motor de plantillas usado por Spring Boot

src/main/resources/
| —— static/
| —— templates/

5.1.1 Carpeta /templates → Vistas HTML (dinámicas)

Spring Boot utiliza Thymeleaf para renderizar archivos HTML que pueden recibir variables enviadas desde los controladores Java.

Página principal y secciones públicas

- index.html → Página de inicio
- galeria.html → Galería de imágenes
- horario.html → Información del horario
- productos.html → Servicios / productos del lavadero
- tarifas.html → Precios
- policy.html → Política de privacidad
- formulario.html → Formulario de reserva/cita
- confirmacion.html → Mensaje de éxito tras enviar formulario
- error.html → Página de error

Subcarpeta /admin → Vistas internas para administradores

Esta carpeta contiene las páginas HTML accesibles únicamente desde la zona privada del sistema de administración. Están diseñadas para personal del lavadero y no para el cliente final.

- listado-citas.html → El administrador puede visualizar todas las citas registradas.
- login.html → Formulario para autenticación del administrador e iniciar sesión
- modelos-vehiculos.html → Página donde se gestionan los modelos de vehículos registrados en el sistema (crear, editar, eliminar).

Subcarpeta /emails → Plantillas para envío de correos HTML

Estas plantillas NO se muestran al usuario directamente, ya que se utiliza para generar correos automáticos enviados al cliente.

- confirmacion-cita.html → Plantilla del correo que recibe el cliente tras reservar una cita. Incluye datos del día, hora, servicio y vehículo.
- Recordatorio-cita.html → Correo automático que se envía antes de la cita como recordatorio.

Subcarpeta /fragments → Elementos reutilizables (Thymeleaf)
Los fragmentos permiten definir piezas de código HTML que luego se pueden incluir en cualquier página.

- cookie-banner.html → Fragmento reutilizable que muestra el aviso de cookies en el footer o en la parte inferior Incluye los botones de aceptar/rechazar.

5.1.2 Carpeta /static → Recursos estáticos

Esta carpeta contiene archivos que NO son procesados por Thymeleaf: son enviados directamente al navegador.

Carpeta /css
Archivos de estilos CSS separados por páginas, dándole mayor modularidad, personalización y adaptabilidad a lo que buscamos:

- cookie-banner.css
- galeria.css
- horario.css
- listado-citas.css
- policy.css
- productos.css
- styles.css (estilos generales)
- tarifas.css

Carpeta /images
Aquí se almacenan:

- Fotos del negocio, Iconos, Imágenes de la galería, Logotipos...

Carpeta /js
Contiene los Scripts encargados de:

- Validación de fecha actual, cargando disponibilidad y limitando horarios.
- Lógica del banner de cookies

5.1.3 Archivo Style.css → Estilos generales

Archivo que nos proporciona los colores, tipografías y estructuras generales de la aplicación y sirve como base para personalización según el lavadero.

Variables Globales (Theme)
El archivo define una serie de variables CSS en :root, que funcionan como la base de colores.

```
:root {
  --color-primary: #0099ff;
  --color-primary-dark: #0d47a1;
  --color-primary-light: #64b5f6;
  --color-background: #f5f7fa;
  --color-text: #333;
  (...)
}
```

Además de una estructura que moldea todas las paginas, proporcionando un diseño responsable y limitando los elementos:

1. Un **reset básico** para normalizar el comportamiento entre navegadores y eliminando espaciados por defecto.

```
{ margin: 0; padding: 0; box-sizing: border-box; }
```

2. El body establece la tipografía y el color general:

```
body {
  font-family: 'Roboto', Arial, sans-serif;
  background: var(--color-background);
  color: var(--color-text);
}
```

3. **La clase .container** se emplea para centrar el contenido y controlar el ancho máximo:

```
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 15px;
}
```

..**El header** usa un fondo primario y flexbox para organizar el logo y el menú:

```
header {
  background: var(--color-primary);
  color: white;
}
```

5. **Los botones:**

```
.btn {
  padding: 8px 16px;
  background: var(--color-primary);
  color: white;
  border-radius: 4px;
  transition: 0.3s;
}
```

6**.Las cards** se utilizan para mostrar módulos y formularios:

```
.card {
  background: white;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  padding: 20px;
}
```

7. Uno de los elementos clave es e**l selector de turismo**, con diseño avanzado:

```
#turismoToggle {
  background: #fff3cd;
  border: 1px solid #ffc107;
  padding: 15px;
}
```

8.El frontend también incluye media para adaptar el diseño a móviles:

```
@media (max-width: 768px) {
  .form-row { flex-direction: column; }
}
```

Para ver vistas Usuario (Visite manual de Uso)

5.2 Aplicación Móvil:

La aplicación móvil del Lavadero está diseñada como un soporte ligero y eficiente para la aplicación de escritorio. Su propósito principal es permitir a los usuarios gestionar citas, visualizar servicios disponibles de lavado y simplificar la comunicación con el backend. Esta app móvil complementa la solución completa del proyecto, proporcionando una alternativa accesible desde cualquier dispositivo Android.

La implementación de esta aplicación se estructura sobre una arquitectura clara y modular, haciendo uso de **Android Jetpack, base de datos local Room y consumo de API REST.**

Diseño de Interfaz

La aplicación móvil tiene una carga visual más simple que la versión de escritorio, buscando mantener una interfaz limpia, clara y fácil de usar, centrada en la funcionalidad esencial: gestión de citas.

XML (Android Layouts)

- FragmentoListaCitas
- FragmentoNuevaCita
- DialogoConfirmacion

- **Archivos clave:**
  - RetrofitClient.kt → crea instancia del cliente HTTP.
  - LavaderoApiService.kt → métodos GET/POST.
  - TipoLavadoRemoto.kt → modelos recibidos del backend.
  - Models.kt → estructura de datos remotos.

Al iniciar la app se muestra el main que refleja el fragmento de NuevaCita, con dos opciones abajo para cambiar entre **NuevaCita y ListaCitas** donde se guardan las citas ya creadas por ti. Al pulsar en la Cita creada dentro de ListaCitas, de despliega el cuadro emergente con más datos y confirmación de la cita.

Para ver vistas Usuario en app móvil (Visite manual de Uso)

6. Conclusiones:

Este proyecto académico ha permitido desarrollar una solución funcional para la gestión de citas de un lavadero, integrando una aplicación móvil ligera y práctica que se comunica con una API expuesta mediante ngrok, lo que facilitó su implementación, pruebas y conectividad entre plataformas durante el proceso de desarrollo.

A pesar de tratarse de un proyecto académico, se logró un producto totalmente operativo que cubre las necesidades fundamentales del proceso de registro y consulta de citas, demostrando una correcta aplicación de conceptos de desarrollo móvil, consumo de servicios REST y estructuración de un flujo de trabajo realista.

El sistema, tal como está implementado, cumple con los objetivos propuestos, pero también deja abierta la puerta a mejoras que permitirían traerlo a un nivel profesional. Entre ellas, destacan la posibilidad de convertir la plataforma en una PWA con capacidades offline, integrar sincronización en tiempo real, añadir paneles de estadísticas, un sistema de notificaciones avanzado, soporte para pagos en línea, además de elementos modernos como modo oscuro, componentes reutilizables, o incluso sistemas de fidelización y análisis financiero.

En conjunto, este proyecto constituye una base eficiente y escalable, demostrando buenas prácticas de diseño y ofreciendo una estructura que puede evolucionar hacia un producto más robusto y completo.

Dificultades encontradas:

- Configuración de ngrok para exponer la API correctamente y mantener la URL actualizada al reiniciar.
- Manejo de errores en la comunicación con la API REST, especialmente ante fallos de red o problemas de formato JSON.
- Creación y sincronización del modelo de datos entre la base de datos local (Room) y el servidor remoto.
- Gestión de estados en la interfaz y comunicación entre ViewModel, repositorio y vistas

Posibles Mejoras:

- **PWA (Progressive Web App)** - Funcionalidad offline
- **Sincronización en tiempo real** (WebSockets)
- **Sistema de notificaciones push** avanzado
- **Integración con pagos online** (Stripe, PayPal)
- **Dashboard con estadísticas** y analytics
- **Historial de clientes** y fidelización
- **API de geolocalización** y mapas
- **Sistema de reviews/calificaciones**
- **Integración con redes sociales**
- **Chatbot de atención al cliente**
- **Sistema de inventario** de productos
- **Reportes financieros** automatizados
- **Animaciones CSS avanzadas** y microinteracciones
- **Tema dark/light mode** para la interfaz
- **Componentes UI reutilizables** con Storybook

7. Bibliografía:

- Ngrok. (2024)
- Android Developers. (2024)
- Oracle. (2024). Documentación oficial de Java SE y JDK.
- MySQL. (2024). Documentación oficial de MySQL y XAMPP.