XCRM - Sistema CRM Multi-Tenant

Documentación Técnica Completa

ÍNDICE

1. Resumen Ejecutivo
2. Introducción
3. Análisis del Entorno
4. Solución Propuesta
5. Tecnologías Utilizadas
6. Planificación Temporal
7. Diseño e Implementación
8. Manual del Usuario
9. Conclusiones
10. Bibliografía

---

1. RESUMEN EJECUTIVO

Español

El proyecto XCRM es una aplicación CRM (Customer Relationship Management) desarrollada con Spring Boot que permite gestionar campañas, clientes, interacciones y generar métricas y reportes. Diseñada para organizaciones que requieren una solución multi-tenant, genera una base de datos por organización al registrarse automáticamente. La aplicación utiliza tecnologías modernas como Spring Security para la autenticación, Thymeleaf para las vistas, Bootstrap para el diseño responsivo y JasperReports para la generación de reportes. El objetivo principal es proporcionar una herramienta eficiente para la gestión de relaciones con los clientes, mejorando la productividad y la toma de decisiones.

English

The XCRM project is a CRM (Customer Relationship Management) application developed with Spring Boot that allows managing campaigns, customers, interactions, and generating metrics and reports. Designed for organizations requiring a multi-tenant solution, it automatically generates a database per organization upon registration. The application uses modern technologies such as Spring Security for authentication, Thymeleaf for views, Bootstrap for responsive design, and JasperReports for report generation. The main objective is to provide an efficient tool for managing customer relationships, improving productivity and decision-making.

---

2. INTRODUCCIÓN

2.1 Contexto del Proyecto

En un entorno empresarial cada vez más competitivo, la gestión eficiente de las relaciones con los clientes se ha convertido en un factor clave para el éxito. Las empresas necesitan herramientas que les permitan organizar y analizar las interacciones con sus clientes de manera efectiva. El proyecto XCRM surge como respuesta a esta necesidad, proporcionando una solución CRM escalable y personalizable.

2.2 Justificación y Relevancia

El desarrollo de XCRM se justifica por la creciente demanda de soluciones CRM que sean:

- Flexibles: Adaptables a diferentes tipos de organizaciones
- Seguras: Con sistemas robustos de autenticación y autorización
- Escalables: Capaces de crecer con las necesidades del negocio
- Multi-tenant: Optimizando recursos y reduciendo costos operativos

2.3 Objetivos Generales y Específicos

Objetivo General: Desarrollar una aplicación CRM multi-tenant que facilite la gestión de campañas, clientes e interacciones, proporcionando herramientas de análisis y reporting para mejorar la toma de decisiones empresariales.

Objetivos Específicos:

1. Implementar autenticación y autorización seguras con Spring Security
2. Diseñar una interfaz intuitiva y responsiva utilizando Thymeleaf y Bootstrap
3. Desarrollar un sistema multi-tenant con bases de datos independientes por organización
4. Generar reportes automatizados en múltiples formatos (PDF, Excel, HTML)
5. Garantizar la escalabilidad del sistema mediante una arquitectura modular
6. Implementar métricas y dashboards para análisis de rendimiento
7. Asegurar la calidad del código mediante pruebas unitarias

2.4 Requisitos del Sistema

Requisitos de Hardware:

- Servidor con mínimo 1 GB de RAM (recomendado 4 GB)
- Procesador con al menos 1 núcleos
- Espacio en disco: 500mb mínimo
- Conexión a internet estable

Requisitos de Software:

- Java 17 o superior
- MySQL 8.0 o MariaDB 10.6+
- Git para control de versiones
- Maven 3.6+ (incluido en el wrapper del proyecto)

2.5 Restricciones y Limitaciones

- Requiere conocimientos técnicos para la configuración inicial
- Limitado a bases de datos MySQL/MariaDB
- El sistema multi-tenant requiere permisos de creación de bases de datos
- Las funcionalidades de reporting requieren JasperReports configurado correctamente

---

3. ANÁLISIS DEL ENTORNO

3.1 Estudio de Viabilidad

Viabilidad Técnica: El proyecto es técnicamente viable utilizando tecnologías maduras y estables como Spring Boot, MySQL y JasperReports. La arquitectura multi-tenant ha sido validada en múltiples proyectos empresariales.

Viabilidad Económica: El uso de tecnologías de código abierto reduce significativamente los costos de licenciamiento. El modelo multi-tenant optimiza el uso de recursos de hardware.

Viabilidad Operacional: La interfaz intuitiva y la documentación completa facilitan la adopción por parte de los usuarios finales.

3.2 Análisis de Requisitos

Requisitos Funcionales:

- RF01: Registro y gestión de organizaciones
- RF02: Sistema de autenticación y autorización por roles
- RF03: Gestión completa de campañas comerciales
- RF04: Registro y seguimiento de clientes
- RF05: Gestión de interacciones cliente-empresa
- RF06: Generación de reportes en múltiples formatos
- RF07: Dashboard con métricas clave
- RF08: Asignación de clientes a campañas
- RF09: Seguimiento de resultados por campaña

Requisitos No Funcionales:

- RNF01: Seguridad - Encriptación de contraseñas y sesiones seguras
- RNF02: Rendimiento - Tiempo de respuesta < 3 segundos
- RNF03: Usabilidad - Interfaz intuitiva y responsiva
- RNF04: Escalabilidad - Soporte para múltiples organizaciones
- RNF05: Disponibilidad - 99.5% de tiempo activo
- RNF06: Mantenibilidad - Código limpio, separación de responsabilidades

3.3 Cronograma del Proyecto:

Fase	Fecha	Dias restantes	Estado
Análisis, Prototipo UI...	24/03/2025	78 días	✅ Completado
Módulo Core Multi-tenant	18/04/2025	53 días	✅ Completado
Dashdoard funcional	20/05/2025	21 días	✅ Completado
Sistema de Reportes	25/05/2025	15 días	✅ Completado
Pruebas y Depuración	28/05/2025	12 días	✅ Completado
Documentación Final	09/06/2025	1 días	⏳ Pendiente
Defensa del Proyecto	10/06/2025	0 día	🎯 Objetivo

3.4 Actores del Sistema

Actor Primario: Administrador de Organización

- Registro de nueva organización
- Configuración del sistema
- Gestión de usuarios
- Acceso completo a reportes y métricas

Actor Secundario: Usuario Comercial

- Gestión de campañas asignadas
- Registro de interacciones con clientes
- Consulta de métricas básicas
- Generación de reportes de sus campañas

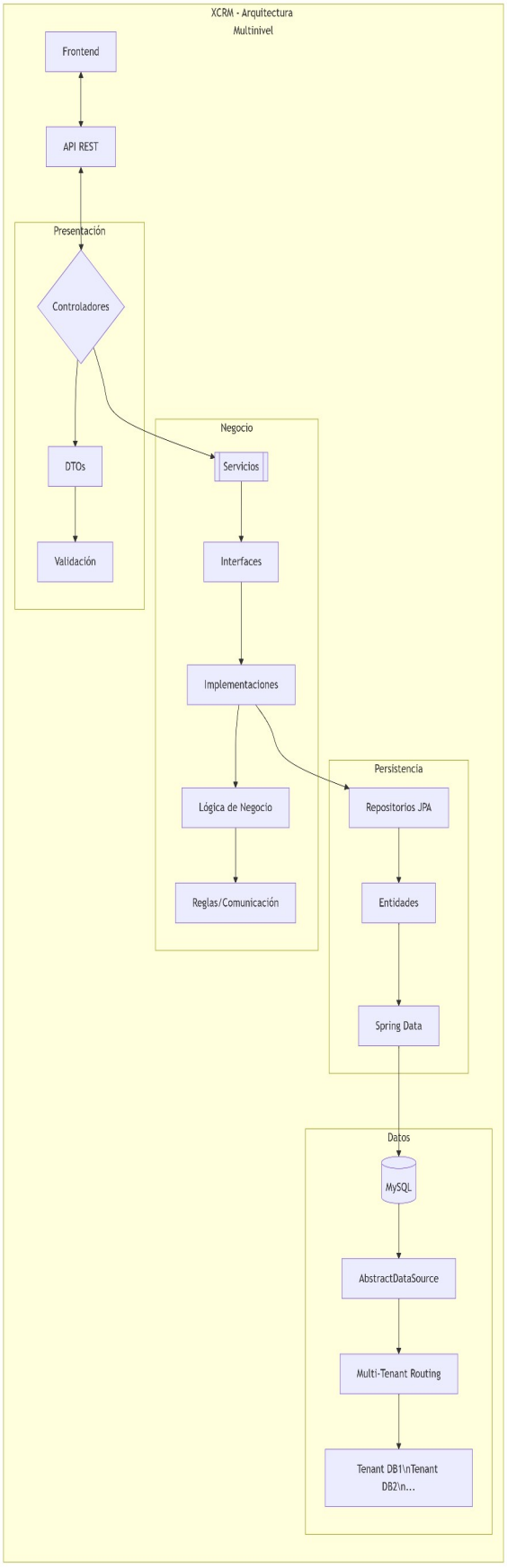
3.5 Análisis de Riesgos

Riesgo	Probabilidad	Impacto	Estrategia de Mitigación
Incompatibilidad entre JasperReports y Spring Boot	Baja	Medio	Pruebas tempranas de integración, prueba de dependencias, pruebas en el servidor.
Sobrecarga de base de datos con múltiples tenants	Media	Alto	Optimización de consultas y índices, uso de AWS RDS, escalar si es necesario
Problemas de seguridad en autenticación	Baja	Muy Alto	Implementación de Spring Security con mejores prácticas, bloqueo por IP

---

4. SOLUCIÓN PROPUESTA

4.1 Arquitectura del Sistema



1. Capa de Presentación:

- ✅ API REST con versionamiento (/api/v1)
- ✅ Controladores con manejo de excepciones
- ✅ DTOs para transferencia de datos (records Java 17+)
- ✅ Validación con Bean Validation 3.0 (@Valid)

•Capa de Negocio:

- ✅ Interfaces/Implementaciones (Principio DIP)
- ✅ Servicios transaccionales (@Transactional)
- ✅ Comunicación asíncrona (Eventos/Spring Integration)

•Capa de Persistencia:

- ✅ Repositorios JPA (Spring Data + Hibernate)
- ✅ Entidades con mapeamiento avanzado (@ManyToMany, @JoinTable)
- ✅ QueryDSL para consultas dinámicas

•Capa de Datos:

- ✅ AbstractDataSource con enrutamiento multi-tenant
- ✅ Connection Pooling (HikariCP)
- ✅ Esquema por Tenant (Strategy Pattern)

Patrones aplicados:

- SOLID (Single Responsibility en servicios)
- Strategy (Multi-Tenancy)
- Factory (DataSource dinámico)

4.2 Modelo Multi-Tenant

Estrategia: Database per Tenant

Cada organización que se registra en el sistema obtiene automáticamente:

- Base de datos dedicada con nombre único
- Esquema completo de tablas
- Configuración de conexión independiente
- Aislamiento completo de datos

Ventajas:

- Máximo aislamiento de datos
- Facilidad para backups individuales
- Escalabilidad independiente por tenant
- Cumplimiento de normativas de privacidad

4.3 Estructura de Bases de Datos

📁 Base de Datos Principal (mi_app)

Contiene información global sobre las organizaciones registradas y sus respectivas configuraciones de conexión.

Tabla: organizaciones

- id (PK): Identificador único de la organización.
- nombre: Nombre de la organización.
- nombreDB: Nombre de la base de datos asociada.
- email: Correo de contacto.
- plan: Tipo de suscripción.
- creado: Fecha de creación.

Tabla: tenant_configs

- organization_id (FK): Clave foránea hacia organizaciones.
- db_host: Host de conexión a la base de datos.
- db_port: Puerto de conexión.
- db_username: Usuario.
- encrypted_password: Contraseña cifrada.

📁 Base de Datos por Tenant

Cada organización tiene su propia base de datos, con tablas específicas para la gestión CRM.

🔐 Seguridad y Autenticación

Tabla: users

- id (PK): UUID del usuario.
- username, password, enabled
- organizacion_id (FK)

- fotoUrl: Foto de perfil (opcional)

Tabla: authorities

- id (PK)
- user_id (FK hacia users)
- authority: Rol asignado al usuario

📊 Gestión de Campañas y Clientes

Tabla: campanias

- id (PK), nombre, descripcion
- fecha_inicio, fecha_fin
- organizacion_id (FK)

Tabla: clientes

- id (PK), nombre, email, telefono
- direccion, organizacion_id (FK)

Tabla: clientes_campanias

- campania_id, cliente_id (PK compuesta)
- Relaciones N:M entre campañas y clientes

Tabla: comerciales_campanias

- campania_id, comercial_id (PK compuesta)
- Asignación de comerciales a campañas

Tabla: comerciales_clientes

- cliente_id, comercial_id (PK compuesta)
- Relación entre comerciales y sus clientes

📞 Interacciones y Ventas

Tabla: interacciones

- id (PK), fecha_hora, estado, tipo, notas
- campania_id, cliente_id, comercial_id (FK)

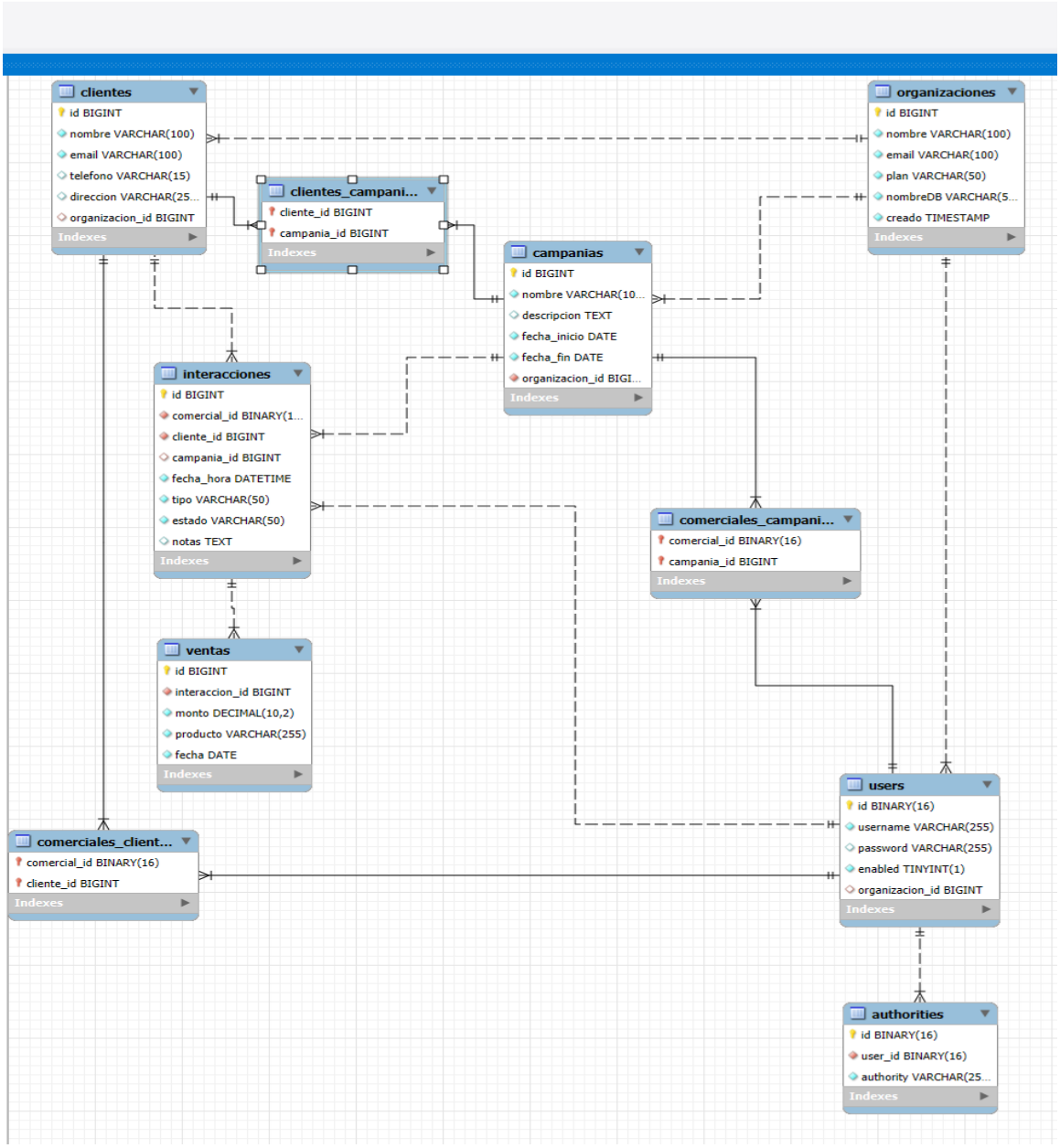
Tabla: ventas

- id (PK), fecha, producto, monto
- interaccion_id (FK, relación 1:1)

📬 Contacto

Tabla: contacto_mensaje

- id (PK), nombre, email, asunto, mensaje
- archivoNombre, archivoDropboxUrl
- fechaEnvio, usuarioId



4.4 Flujo de Trabajo

1. Registro e Inicialización de la Organización

- Usuario → Formulario de Registro
- Se recogen los datos básicos de la organización y del usuario administrador.
- Validación de datos
- Creación de la Organización y su Base de Datos
- Se genera automáticamente una base de datos exclusiva para la organización.
- Creación del Usuario Administrador
- Este usuario tiene acceso total para gestionar el entorno de su organización.

- Configuración Inicial del Sistema
- Se cargan configuraciones base y se habilita el acceso a los módulos.

---

2. Gestión Administrativa y Operativa

Accesible tras el login del usuario administrador:

- Usuarios
- Creación y gestión de cuentas internas (comerciales, analistas...).
- Asignación de roles y permisos.
- Clientes
- Registro y mantenimiento de información comercial de los clientes.
- Posibilidad de asignar clientes a campañas y comerciales.
- Campañas
- Creación de campañas de marketing/ventas.
- Asignación de campañas a usuarios comerciales.
- Asociación opcional de clientes a campañas específicas.

---

3. Interacciones Comerciales

- Selección de Campaña y Cliente
- El comercial accede a las campañas asignadas.
- Puede elegir un cliente para interactuar.
- Registro de Interacción
- Modalidad: llamada, email, mensaje, visita presencial.
- Estado: exitosa, en proceso, no contactado, etc.
- Detalles: observaciones, resultados y comentarios.
- Historial de Interacciones
- Visualización del histórico completo de interacciones por cliente.
- Incluye todas las campañas y comerciales involucrados.

---

4. Ventas

- Acceso desde la sección de Ventas
- Selección de campaña y cliente.
- Vinculación directa con una interacción previa.
- Registro del producto vendido y el monto.
- Análisis de Rendimiento

- Cada venta queda asociada a una interacción y alimenta los reportes.

---

5. Dashboard y Métricas

- Visualización de KPI
- Uso de gráficos generados con Chart.js y Jasper Report
- Métricas de rendimiento: interacciones, campañas activas, tasas de conversión, etc.
- Reportes Exportables
- Descarga de informes PDF generados con JasperReports.
- Reportes disponibles:
- Interacciones por comercial.
- Todos los clientes.
- Todos los usuarios de la organización.

4.5 Seguridad Implementada

La capa de seguridad del sistema ha sido diseñada siguiendo buenas prácticas de desarrollo seguro, garantizando la integridad, confidencialidad y disponibilidad de los recursos y datos sensibles. A continuación se detalla cada uno de los componentes implementados:

Autenticación:

- Formularios de autenticación personalizados

Se utiliza Spring Security para gestionar la autenticación mediante formularios web configurados específicamente para las necesidades del proyecto.

- Encriptación de contraseñas con BCryptPasswordEncoder

Todas las contraseñas son almacenadas utilizando un algoritmo de encriptación robusto y seguro basado en BCrypt, evitando riesgos asociados al almacenamiento en texto plano o en formatos débiles.

- Gestión segura de sesiones

Las sesiones son manejadas de forma segura, incluyendo la invalidación automática tras cerrar sesión y la protección contra fijación de sesión (session fixation protection).

- Logout automático por inactividad

El sistema invalida automáticamente las sesiones de usuario tras un periodo de inactividad prolongado, mejorando la seguridad en entornos compartidos.

Autorización:

- Control de acceso basado en roles (RBAC)

El acceso a funcionalidades críticas del sistema está restringido según el rol del usuario (por ejemplo: ADMIN, USER). Esto se implementa a través de anotaciones como @PreAuthorize y configuraciones en las rutas.

- Filtros de seguridad por URL

Se han definido permisos granulares a nivel de URL, permitiendo acceso público solo a ciertos recursos y protegiendo el resto bajo autenticación y autorización.

- Validación de permisos a nivel de método

En operaciones críticas del backend se utilizan anotaciones como @PreAuthorize("hasRole('ADMIN')") para garantizar que únicamente usuarios autorizados puedan ejecutar dichas acciones.

- Protección CSRF activa

La protección CSRF está habilitada globalmente, salvo en endpoints explícitamente definidos como seguros para no requerirla (por ejemplo, APIs REST externas).

Protección de Datos:

- Aislamiento multi-tenant

El sistema soporta arquitectura multi-inquilino gracias a la clase CustomRoutingDataSource, que selecciona dinámicamente la base de datos según el tenant actual, asegurando así el aislamiento de datos entre clientes.

- Validación de inputs

Todos los campos recibidos desde el frontend son validados antes de ser procesados, mitigando riesgos como inyección SQL o XSS.

- Registro (logging) de accesos y cambios

Se registran todos los eventos relevantes como logins exitosos/fallidos, cambios en configuraciones críticas y accesos a recursos sensibles, facilitando auditorías futuras.

Limitación de Tasa (Rate Limiting)

- Protección de rutas POST críticas con Bucket4J

Para prevenir ataques de fuerza bruta y abuso de servicios públicos, se ha implementado una política de limitación de intentos por dirección IP en las siguientes rutas:

- /login

- /registro

- /enviar-contacto

- Configuración del límite de intentos

rate.limit.routes=/login,/registro,/enviar-contacto

rate.limit.max-attempts=5

rate.limit.refill-duration=60

Esto significa que si una IP realiza más de 5 solicitudes en 60 segundos, será bloqueada temporalmente hasta que se reinicie su cuota.

- Filtro personalizado: LoginRateLimitingFilter

Este filtro intercepta las peticiones a las rutas protegidas y aplica la lógica de control de tasa, devolviendo una respuesta HTTP 429 (Too Many Requests) cuando se supera el umbral permitido.

Integración en Spring Security:

El sistema de seguridad está completamente integrado con Spring Security mediante la clase SecurityConfig, donde se define:

- Configuración de URLs públicas y privadas.

- Personalización del formulario de login.

- Manejo del logout, incluyendo limpieza de cookies y reseteo de contexto de datos.

- Inclusión del filtro personalizado de rate limiting antes del filtro estándar de autenticación (UsernamePasswordAuthenticationFilter).

- Ignorar protección CSRF en endpoints específicos como APIs REST o reportes.

5. TECNOLOGÍAS UTILIZADAS

5.1 Backend: Spring Boot

Se ha utilizado Spring Boot 3.3.5 como framework principal del backend, lo que permite desarrollar aplicaciones empresariales en Java con una configuración mínima y una estructura modular. Entre sus características destacadas se encuentran:

- Autoconfiguración inteligente mediante convenciones.
- Servidor embebido Tomcat que facilita el despliegue sin necesidad de contenedores externos.
- Gestión optimizada de dependencias a través del spring-boot-starter-parent.
- Soporte para múltiples perfiles de entorno (dev, prod), lo que permite una configuración flexible.

Módulos de Spring empleados:

- Spring MVC: Implementación de controladores web y APIs REST.
- Spring Security 6: Gestión de autenticación y autorización basada en roles.
- Spring Data JPA: Acceso a datos mediante repositorios sobre entidades persistentes.

- Spring Boot Actuator: Exposición de métricas, salud y trazabilidad.
- Spring Boot Mail: Envío de correos automáticos mediante SMTP.
- Spring Cache + Caffeine: Mejora del rendimiento mediante almacenamiento en memoria.

El proyecto utiliza Java 17 como versión del JDK, garantizando compatibilidad con las últimas funcionalidades del lenguaje.

5.2 Frontend: Thymeleaf + Bootstrap

El motor de plantillas utilizado es Thymeleaf 3.1.x, integrado con Spring Boot para la generación de contenido HTML desde el servidor. Permite una sintaxis natural basada en HTML5 y una estructura modular mediante fragmentos reutilizables. También ofrece compatibilidad con Spring Security para mostrar o ocultar contenido dinámico según los permisos del usuario.

Para el diseño visual se ha empleado Bootstrap 5.3.x, que proporciona:

- Un sistema de rejilla responsiva con enfoque mobile-first.
- Componentes predefinidos como formularios, botones, alertas y navegación.
- Estilos personalizables y adaptación al branding del sistema.
- Iconografía mediante Bootstrap Icons.

JavaScript nativo ha sido utilizado para complementar la experiencia del usuario, implementando validaciones en tiempo real, peticiones AJAX y animaciones con la librería AOS (Animate On Scroll).

5.3 Persistencia de Datos: MySQL y Multitenancy

La base de datos principal del sistema es MySQL 8.0+, por su alto rendimiento en aplicaciones web, compatibilidad con MariaDB y facilidad de administración. Se han aplicado buenas prácticas de diseño de esquemas y uso de índices para mejorar el acceso concurrente.

El sistema implementa una arquitectura multi-tenant con las siguientes características:

- Separación de datos por organización mediante bases de datos independientes.
- Conmutación dinámica de fuentes de datos (DataSource) usando un AbstractRoutingDataSource.
- HikariCP como gestor de conexiones de alto rendimiento.
- Gestión automática de transacciones y aislamiento de sesiones.

5.4 Generación de Informes: JasperReports

Para la generación de reportes se ha integrado JasperReports 7.0.2, un motor potente para la creación de informes empresariales.

Se utiliza JasperSoft Studio como entorno de diseño visual, con soporte para:

- Exportación en múltiples formatos: PDF, Excel, HTML.
- Visualizaciones mediante gráficos, tablas dinámicas y componentes personalizados.
- Integración directa con Spring Boot mediante servicios y beans configurados.

Funcionalidades clave implementadas:

- Informes detallados por cliente, campaña y comercial.
- Métricas sobre rendimiento comercial, interacción y conversión.
- Dashboards interactivos con filtros y exportación.
- Automatización de reportes mediante programación periódica.

5.5 Control de Versiones: Git

El control de versiones se gestiona mediante Git, siguiendo el modelo de ramas Git Flow, que permite un desarrollo organizado y colaborativo.

Las ramas utilizadas son:

- main: Rama principal con el código listo para producción.
- develop: Rama de integración de nuevas funcionalidades.
- feature/*: Ramas específicas para cada nueva característica.

Esta metodología permite mantener una trazabilidad clara del proyecto, favorece la colaboración en equipo y reduce errores en el proceso de despliegue.

---

6. PLANIFICACIÓN TEMPORAL

6.1 Metodología Seleccionada

- Revisiones regulares cada 2 viernes con el tutor y entre los desarrolladores
- Entrega continua con subida a feature → develop constante.

6.2 Fases del Desarrollo

Fase 1: Investigación y Análisis (4 semanas)

- Estudio de tecnologías y frameworks
- Definición de requisitos detallados
- Diseño de arquitectura multi-tenant
- Configuración del entorno de desarrollo

Fase 2: Desarrollo Core (6 semanas)

- Implementación de la base multi-tenant
- Sistema de autenticación y autorización
- Modelos de datos y repositorios
- APIs básicas de gestión

Fase 3: Funcionalidades CRM (4 semanas)

- Gestión de campañas y clientes
- Sistema de interacciones
- Dashboard y métricas básicas
- Integración frontend-backend

Fase 4: Sistema de Reportes (3 semanas)

- Configuración de JasperReports
- Diseño de plantillas de reportes
- Generación automática de PDFs

- Exportación a múltiples formatos

Fase 5: Testing y Refinamiento (3 semanas)

- Pruebas unitarias e integración
- Testing de usabilidad
- Optimización de rendimiento con cacheo y librerias no bloqueantes
- Corrección de bugs

6.3 Recursos Necesarios

Equipo de Desarrollo

- Desarrolladores Full-Stack : Profesionales con experiencia en tecnologías de backend (Java, Spring Boot) y frontend (Thymeleaf, Bootstrap), responsables del desarrollo integral del sistema.

Herramientas y Software

Entornos de Desarrollo (IDE)

- IntelliJ IDEA : Utilizado para la escritura y depuración del código Java y configuración de Spring Boot.

Bases de Datos

- MySQL Workbench : Para modelado, gestión y pruebas sobre la base de datos.
- MariaDB : Alternativa compatible con MySQL utilizada en entornos locales o pruebas.

Diseño y Reporting

- JasperSoft Studio : Creación y edición visual de informes en JasperReports.
- Software de edición gráfica y textual (por ejemplo: GIMP, LibreOffice, Inkscape): Para recursos visuales y documentación.

Control de Versiones

- Git : Sistema de control de versiones distribuido.
- GitHub : Repositorio central y plataforma de colaboración.
- Fork Client : Cliente de escritorio para gestión visual de ramas y commits.

Infraestructura Cloud

- Amazon Web Services (AWS) :
    - EC2 (Elastic Compute Cloud) : Para el despliegue del servidor de aplicación (Spring Boot + Tomcat).
    - RDS (Relational Database Service) : Para alojar las bases de datos MySQL de producción, incluyendo el soporte multitenant.
    - Opcionalmente , se pueden utilizar servicios complementarios como:
        - S3 : Almacenamiento de archivos y respaldos.
        - CloudWatch : Monitorización del sistema en producción.

7. DISEÑO E IMPLEMENTACIÓN

7.1 Diagrama de Clases

Entidades Principales:

```
@Entity
public class Organization {
   @Id @GeneratedValue
   private Long id;
```

```
    private String name;
    private String databaseName;
    private LocalDateTime createdAt;
    private OrganizationStatus status;
}

@Entity
public class Campaign {
    @Id @GeneratedValue
    private Long id;
    private String name;
    private String description;
    private LocalDate startDate;
    private LocalDate endDate;
    private CampaignStatus status;

    @ManyToOne
    private User createdBy;

    @OneToMany(mappedBy = "campaign")
    private List<Interaction> interactions;
}

@Entity
public class Client {
    @Id @GeneratedValue
    private Long id;
    private String name;
    private String email;
    private String phone;
    private String company;
    private String address;

    @OneToMany(mappedBy = "client")
    private List<Interaction> interactions;
}

@Entity
public class Interaction {
    @Id @GeneratedValue
    private Long id;

    @ManyToOne
    private Campaign campaign;

    @ManyToOne
    private Client client;

    private InteractionType type;
    private InteractionStatus status;
    private LocalDateTime interactionDate;
    private String notes;
    private String result;
}
```

7.2 Estructura del Código

Organización de Paquetes:

```
com.xcrm/
├── configuration/          # Configuraciones de Spring
│   ├── SecurityConfig.java
│   ├── MultiTenantConfig.java
│   └── JasperConfig.java
├── controller/
│   ├── web/                # Controladores MVC
│   │   ├── HomeController.java
│   │   ├── CampaignController.java
│   │   └── ReportController.java
│   └── rest/               # APIs REST
│       ├── CampaignRestController.java
│       └── MetricsRestController.java
├── dto/                    # Objetos de transferencia
│   ├── CampaignDTO.java
│   ├── ClientDTO.java
│   └── InteractionDTO.java
├── model/                  # Entidades JPA
│   ├── Organization.java
│   ├── Campaign.java
│   ├── Client.java
│   └── Interaction.java
├── repository/             # Repositorios de datos
│   ├── CampaignRepository.java
│   ├── ClientRepository.java
│   └── InteractionRepository.java
├── service/                # Lógica de negocio
│   ├── CampaignService.java
│   ├── ClientService.java
│   ├── ReportService.java
│   └── TenantService.java
└── utils/                  # Utilidades
    ├── DateUtils.java
    ├── SecurityUtils.java
    └── TenantContext.java
```

7.3 Patrones de Diseño Aplicados

1. Repository Pattern

```
@Repository
public interface CampaignRepository extends JpaRepository<Campaign, Long> {
    List<Campaign> findByStatusAndCreatedBy(CampaignStatus status, User user);

    @Query("SELECT c FROM Campaign c WHERE c.endDate < :date")
    List<Campaign> findExpiredCampaigns(@Param("date") LocalDate date);
}
```

2. Service Layer Pattern

```
@Service
@Transactional
public class CampaignService {

    private final CampaignRepository campaignRepository;
    private final InteractionRepository interactionRepository;

    public CampaignDTO createCampaign(CampaignDTO dto) {
        // Lógica de negocio
        Campaign campaign = convertToEntity(dto);
        campaign = campaignRepository.save(campaign);
        return convertToDTO(campaign);
```

```
    }
}
```

3. DTO Pattern

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CampaignDTO {
    private Long id;
    private String name;
    private String description;
    private LocalDate startDate;
    private LocalDate endDate;
    private String status;
    private List<InteractionDTO> interactions;
}
```

4. Factory Pattern (Multi-Tenant)

```
@Component
public class TenantDataSourceFactory {

    public DataSource createDataSource(String tenantId) {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://localhost:3306/" + tenantId);
        config.setUsername(getUsername(tenantId));
        config.setPassword(getPassword(tenantId));
        return new HikariDataSource(config);
    }
}
```

7.4 Casos de Uso Implementados

CU01: Registro de Organización

```
@PostMapping("/register")
public String registerOrganization(@Valid OrganizationDTO dto,
                                   BindingResult result) {
    if (result.hasErrors()) {
        return "register";
    }

    try {
        organizationService.createOrganization(dto);
        return "redirect:/login?registered";
    } catch (Exception e) {
        result.rejectValue("name", "error.organization",
                           "Error creating organization");
        return "register";
    }
}
```

CU02: Gestión de Campañas

```
@PostMapping("/campaigns")
public String createCampaign(@Valid CampaignDTO dto,
                             BindingResult result) {
    if (result.hasErrors()) {
        return "campaigns/form";
    }

    campaignService.createCampaign(dto);
```

```
    return "redirect:/campaigns?success";
}
```

CU03: Generación de Reportes

```
@GetMapping("/reports/campaign/{id}")
public ResponseEntity<byte[]> generateCampaignReport(@PathVariable Long id,
                                                     @RequestParam String format) {
    try {
        byte[] report = reportService.generateCampaignReport(id, format);

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(getMediaType(format));
        headers.setContentDispositionFormData("attachment",
                                              "campaign_report." + format);

        return ResponseEntity.ok()
                .headers(headers)
                .body(report);
    } catch (Exception e) {
        return ResponseEntity.status(500).build();
    }
}
```

7.5 API Desarrollada

🔧 Actuator / Monitoring

Método	Ruta	Descripción
GET	/actuator	Muestra los endpoints disponibles
GET	/actuator/health	Estado de salud del sistema
GET	/actuator/health/**	Detalles extendidos del estado
GET	/actuator/mappings	Lista todas las rutas mapeadas
GET	/actuator/info	Información general del sistema

🏠 Páginas Públicas

Método	Ruta	Descripción
GET	/	Página principal
GET	/caracteristicas	Características del producto
GET	/precios	Información sobre precios
GET	/contacto	Formulario de contacto
POST	/enviar-contacto	Procesa el formulario de contacto
GET	/aviso-legal	Aviso legal del sitio
GET	/login	Formulario de inicio de sesión
GET	/registro	Formulario de registro
POST	/registro	Procesa el registro de nuevos usuarios

👤 Gestión de Usuarios

Método	Ruta	Descripción
GET	/usuarios/administration	Panel de administración de usuarios
GET	/usuarios/configuracion	Configuración del usuario actual
GET	/usuarios/usuarios/configuracion	Configuración avanzada de usuarios

Método	Ruta	Descripción
POST	/usuarios/actualizar-configuracion	Actualiza configuraciones del usuario
POST	/usuarios/registrar	Registro de nuevo usuario
POST	/usuarios/editar	Edita datos de un usuario existente
POST	/usuarios/eliminar	Elimina un usuario del sistema
GET	/usuarios/edit-mi-perfil	Edición del perfil del usuario actual
POST	/usuarios/edit-mi-perfil/update	Guarda cambios del perfil del usuario
POST	/usuarios/guardar-foto	Sube o actualiza foto de perfil
POST	/usuarios/enviar-ticket-edicion	Envía ticket para edición de datos

---

🎯 Campañas (Campaigns)

Método	Ruta	Descripción
GET	/campaigns/administration	Panel de administración de campañas
GET	/campaigns/edit/{id}	Formulario de edición de campaña
POST	/campaigns/update	Actualiza los datos de una campaña
POST	/campaigns/delete	Elimina una campaña
POST	/campaigns	Crea una nueva campaña

💼 Ventas / Comercial

Método	Ruta	Descripción
GET	/sales	Dashboard de ventas/comercial
GET	/sales/clientByIdCampaign	Clientes asociados a una campaña
GET	/sales/interaction/{id}	Formulario para registrar interacción
POST	/sales/interaction/save	Guarda una interacción registrada
POST	/sales/assign	Asigna campañas a usuarios comerciales

📊 Reportes

Método	Ruta	Descripción
GET	/report	Dashboard de reportes
GET	/report/admin/ventas	Reporte de ventas por comercial
GET	/report/admin/interacionesPorVenta	Reporte de interacciones por venta
POST	/report/generate	Genera un informe personalizado

📋 Clientes

Método	Ruta	Descripción
GET	/clients	Dashboard de gestión de clientes
POST	/clients	Registra un nuevo cliente
GET	/clients/edit	Muestra formulario de edición de cliente
POST	/clients/update	Actualiza los datos de un cliente
POST	/clients/delete	Elimina un cliente

⚙️ Configuración y Administración

Método	Ruta	Descripción
POST	/api/cache/clear	Limpia la caché del sistema
GET	/api/config/export-db	Exporta la base de datos del sistema

---

📈 API REST - Reportes

Método	Ruta	Descripción
GET	/api/ventas-por-comercial	Datos de ventas agrupados por comercial
GET	/api/ventas-por-campania	Datos de ventas agrupados por campaña
GET	/api/interacciones-por-venta	Datos de interacciones por venta

---

🚫 Errores y Recursos Estáticos

Método	Ruta	Descripción
Todos	/error	Manejador genérico de errores HTTP
GET	/webjars/**	Acceso a recursos estáticos (WebJars)
GET	/**	Acceso a archivos estáticos generales
GET	/uploads/**	Archivos subidos por usuarios (ej: imágenes)

---

8. MANUAL DEL USUARIO

8.1 Instalación y Configuración

Prerrequisitos del Sistema

Antes de instalar XCRM, asegúrate de tener:

Software Requerido:

- Java JDK 17 o superior
- MySQL 8.0 o MariaDB 10.6+
- Git (para clonar el repositorio)

Verificación de Java:

```
java -version
# Debe mostrar: openjdk version "17.0.x" o superior
```

Verificación de MySQL:

```
mysql --version
# Debe mostrar: mysql Ver 8.0.x o superior
```

Paso 1: Clonar el Repositorio

```
# Clonar desde GitHub
git clone https://github.com/True1Santony/xcrm.git
cd xcrm

# Cambiar a la rama de desarrollo (opcional)
git checkout develop
```

Paso 2: Configuración de Base de Datos

Crear base de datos principal:

```
-- Conectarse a MySQL como root
mysql -u root -p

-- Crear la base de datos principal
CREATE DATABASE mi_app CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Crear usuario para la aplicación (opcional pero recomendado)
CREATE USER 'xcrm_user'@'localhost' IDENTIFIED BY 'xcrm_password_2025';
GRANT ALL PRIVILEGES ON mi_app.* TO 'xcrm_user'@'localhost';
GRANT CREATE ON *.* TO 'xcrm_user'@'localhost';
FLUSH PRIVILEGES;

EXIT;
```

Paso 3: Configuración de la Aplicación

Editar archivo de configuración:

```
# src/main/resources/application.properties

# Configuración de la base de datos principal
spring.datasource.url=jdbc:mysql://localhost:3306/mi_app?useSSL=false&serverTimezone=UTC
spring.datasource.username=xcrm_user
spring.datasource.password=xcrm_password_2025
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Configuración JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# Configuración del servidor
server.port=8080

# Configuración de logging
logging.level.com.xcrm=DEBUG
logging.file.name=logs/xcrm.log
```

Paso 4: Compilación y Ejecución

Usando Maven Wrapper (recomendado):

```
# En sistemas Unix/Linux/Mac
./mvnw clean install
./mvnw spring-boot:run

# En Windows
mvnw.cmd clean install
mvnw.cmd spring-boot:run
```

Usando Maven instalado:

```
mvn clean install
mvn spring-boot:run
```

Paso 5: Verificación de la Instalación

Acceder a la aplicación:

URL: http://localhost:8080

Página de inicio esperada:

- intex.html

Aquí tienes las secciones desarrolladas de **Conclusiones** y **Bibliografía** para tu documentación técnica de XCRM:

---

9. CONCLUSIONES

El desarrollo del sistema XCRM ha permitido implementar una solución CRM multi-tenant moderna, segura y escalable, orientada a satisfacer las necesidades de organizaciones que gestionan campañas comerciales, interacciones y ventas. A lo largo del proyecto se han abordado desafíos técnicos significativos como la arquitectura multi-tenant basada en bases de datos por organización, la integración de JasperReports para generación de informes, y la configuración de seguridad con Spring Security, obteniendo resultados satisfactorios.

Entre los principales logros destacan:

- **Aislamiento total de datos entre organizaciones**, permitiendo cumplir con normativas de privacidad y facilitar tareas de mantenimiento como copias de seguridad por tenant.
- **Automatización en la creación de organizaciones y bases de datos**, reduciendo la fricción en el alta de nuevos clientes.
- **Generación de informes en múltiples formatos**, mejorando el análisis de rendimiento para toma de decisiones empresariales.
- **Interfaz amigable y responsiva** con Bootstrap y Thymeleaf, favoreciendo la experiencia de usuario.
- **Seguridad reforzada**, con autenticación encriptada, control de sesiones y limitación de intentos para rutas críticas.

El proyecto ha seguido una planificación ordenada por fases, cumpliendo los plazos estimados y validando cada módulo mediante pruebas unitarias y de integración. Gracias al uso de tecnologías robustas y al enfoque modular, XCRM se encuentra preparado para escalar y adaptarse a entornos reales.

Como mejora futura, se propone:

- Incorporar un sistema de notificaciones SMS.
- Permitir subir documentación de la organización, como explicación detallada de campañas, procedimientos.
- Importación/exportación de db mas detallada, no globalmente.
- Añadir un módulo de análisis predictivo con aprendizaje automático.
- Chatbot LLM, por API IA.
- Desplegar versiones móviles nativas para Android/iOS.
- Mejora de Arquitectura a Micro servicios distribuidos, si no fuera suficiente el escalado vertical.

XCRM no solo cumple los requisitos funcionales y no funcionales establecidos, sino que sienta una base sólida para la evolución futura del sistema.

10. BIBLIOGRAFÍA

- Spring Boot Official Documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/
- Spring Security Reference. https://docs.spring.io/spring-security/reference/index.html
- JasperReports Library. https://community.jaspersoft.com/project/jasperreports-library
- Thymeleaf Official Guide. https://www.thymeleaf.org/documentation.html
- Bootstrap Documentation. https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Java SE 17 Documentation. https://docs.oracle.com/en/java/javase/17/
- MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/
- QueryDSL Reference Guide. http://www.querydsl.com/static/querydsl/latest/reference/html_single/
- Git Flow – A successful Git branching model. https://nvie.com/posts/a-successful-git-branching-model/
- OWASP Foundation – Top 10 Web Application Security Risks. https://owasp.org/www-project-top-ten/