5.Arquitectura

Nuestro objetivo es construir una solución simple pero robusta con una arquitectura minimalista que permita la entrega rápida de valor y mantenga la simplicidad operativa. Todo el sistema funciona localmente utilizando Docker Compose, lo que garantiza un entorno de desarrollo consistente y fácil de replicar.

5.1. Diagrama (C1/C2: Contexto y Contenedores)

El sistema se compone de tres contenedores principales orquestados por Docker Compose y la aplicación cliente. La API REST Spring Boot actúa como un monolito manejando toda la lógica de negocio y accediendo directamente a MySQL mediante JDBC.

- Usuario: Interactúa con la aplicación.
- App Flutter/Android: Interfaz de usuario nativa encargada de realizar llamadas REST sobre HTTPS + JSON a la API.
- API REST Spring Boot Monolito: Expone los endpoints y contiene la lógica de negocio (validación de citas).
- MySQL: Base de datos relacional para almacenar citas, profesionales y servicios.
- Adminer (Opcional Dev): Herramienta para visualizar y editar la base de datos durante el desarrollo.

5.2. Decisiones de Arquitectura (ADR)

Hemos definido 10 ADRs fundamentales para guiar el desarrollo , garantizando la coherencia arquitectónica y proporcionando pasos concretos para la implementación.

5.2.1. ADR-001: Arquitectura Monolítica con Spring Boot

- Decisión: Construir un único proyecto Spring Boot que expone endpoints REST y maneja todas las operaciones en un solo artefacto desplegable.
- Motivo: Es la opción más simple de desplegar, entender y mantener para un equipo pequeño de DAM. Este enfoque no impide una futura migración a microservicios.
- Implementación (Ejemplo): Organizar el código en paquetes claros: controller/, service/, dao/, model/ y dto/.

5.2.2. ADR-002: Seguridad Simplificada con API Key

- Decisión: Para la Iteración 1, usar un sistema de autenticación simple basado en una clave de API fija (X-API-KEY).

- Motivo: Evita el bloqueo por la complejidad de implementar JWT completo, permitiendo dedicar el tiempo a la funcionalidad core (Reserva de Citas, HU-005).
- Implementación (Ejemplo): Implementar un filtro servlet que valide la cabecera X-API-KEY y retornar código HTTP 401 si la clave es inválida.

5.2.3. ADR-003: Acceso a Datos con Spring JDBC Template

- Decisión: Utilizar Spring JDBC Template para el acceso a datos en lugar de JPA/Hibernate.
- Motivo: Permite escribir SQL explícito , lo que es sencillo para equipos en formación y evita consultas ineficientes generadas automáticamente.
- Implementación (Ejemplo): Crear clases DAO (Data Access Object) con métodos CRUD y usar SQL parametrizado para prevenir inyección SQL.

5.2.4. ADR-004: Versionado de Esquema con Flyway

- Decisión: Implementar Flyway para la migración de la base de datos.
- Motivo: Garantiza que todos los miembros del equipo trabajen con exactamente la misma estructura de base de datos, eliminando problemas de desincronización.
- Implementación (Ejemplo): Documentar los cambios de esquema en archivos SQL secuenciales ubicados en src/main/resources/db/migration/.

5.2.5. ADR-010: Despliegue Local con Docker Compose

- Decisión: Crear un archivo docker-compose.yml para orquestar los servicios MySQL, la API Spring Boot y, opcionalmente, Adminer.
- Motivo: Permite levantar todo el stack completo con un solo comando (docker-compose up) , asegurando un entorno de desarrollo idéntico para todos.
- Implementación (Ejemplo): Configurar el servicio MySQL con un volumen persistente para los datos y utilizar un usuario app_user con permisos limitado

5.3. Integraciones, Datos y Dependencias

- Tecnologías de Stack: Spring Boot (Backend), MySQL (BD) , App Flutter (Frontend).
- Datos Sensibles: Las credenciales (contraseñas de BD, API Key) se almacenarán en variables de entorno (archivo .env), nunca en el código fuente.
- Versionado de API: Todas las rutas usarán el prefijo /api/v1/ para permitir futuras evoluciones sin romper clientes existentes.

Riesgo Técnico	Impacto	Mitigación
Desincronización de BD	Fallos en el arranque o errores de SQL en entornos de desarrollo.	Uso obligatorio de **Flyway (ADR-004)** para aplicar migraciones automáticamente al inicio.
Complejidad de Seguridad	Bloqueo en la implementación de la funcionalidad *core*.	Uso de **API Key fija (ADR-002)** para simplificar la seguridad inicial.
Inyección SQL	Vulnerabilidad de la aplicación debido al uso de SQL explícito.	Uso de **SQL parametrizado** en todos los métodos del DAO con Spring JDBC Template (ADR-003).
Exposición de Credenciales	Fuga de datos sensibles al subir a repositorios públicos.	Uso estricto de **Variables de Entorno** (ADR-008) y no registrar contraseñas/tokens en los logs (ADR-009).