Documentación Técnica - BabyTrackMaster

# 1. Arquitectura General

La aplicación BabyTrackMaster está basada en una arquitectura de microservicios desacoplada, compuesta por servicios backend desarrollados con Spring Boot y un frontend desarrollado en React.

La infraestructura se gestiona con Docker y Docker Compose, mientras que Jenkins se encarga de la integración y entrega continua (CI/CD).

# 2. Tecnologías Utilizadas

▪ Backend: Java 17, Spring Boot 3.x, Spring Security, Spring Data JPA, Spring Cloud Config, Swagger, JWT.

▪ Frontend: React, Material UI, React Router, Axios.

▪ Base de datos: MySQL (una por microservicio).

▪ Infraestructura: Docker, Docker Compose, Jenkins.

▪ Otros: Prometheus, Grafana, ELK Stack, Amazon S3 / Cloudinary, Mailtrap / SMTP.

# 3. Microservicios Backend

▪ api-bff: servicio Backend for Frontend que actúa como orquestador para el Dashboard General de Bienvenida.   
 Centraliza y unifica las llamadas a múltiples microservicios (api-cuidados, api-citas, api-rutinas, profile-service)   
 proporcionando un único endpoint optimizado que devuelve rutinas del día, próximas citas, últimos cuidados y estadísticas rápidas.  
 Mejora el rendimiento reduciendo la latencia y simplificando la lógica en el frontend.

▪ api-usuarios: gestión de usuarios y autenticación con JWT.

▪ api-cuidados: registro de biberones, pañales, sueño, baño.

▪ api-gastos: control de gastos por categoría y mes.

▪ api-hitos: registro cronológico de hitos con posibilidad de imágenes.

▪ api-citas: control de citas médicas y notificaciones por email.

▪ api-diario: entradas personales con texto, emociones e imágenes.

▪ api-rutinas: planificación de actividades semanales del bebé.

▪ config-server: distribución centralizada de configuración.

▪ api-gateway: enrutamiento de peticiones y filtrado JWT.

# 4. Estructura del Frontend (React)

▪ Proyecto creado con Create React App.

▪ Ruteo gestionado con React Router.

▪ Componentes desacoplados por módulo: cuidados, gastos, diario, citas, rutinas, etc.

▪ Axios para consumo de APIs con token JWT en headers.

▪ Almacenamiento de sesión en localStorage.

▪ Gestión de estado local con Context o useReducer.

# 5. Seguridad

▪ Spring Security configurado en cada microservicio con SecurityFilterChain.

▪ Tokens JWT firmados y con expiración.

▪ Autenticación y autorización gestionadas por roles.

▪ Endpoints públicos: /auth/register, /auth/login.

▪ Endpoints protegidos por JWT y filtros personalizados.

# 6. Base de Datos

▪ Cada microservicio posee su propia base de datos MySQL.

▪ Tablas diseñadas según el dominio funcional (por ejemplo: Gasto, EntradaDiario, Rutina).

▪ Integración con Spring Data JPA (repositorios).

▪ Configuración de conexión vía application.yml (o bootstrap.yml desde config-server).

# 7. Docker y Docker Compose

▪ Cada microservicio contiene su propio Dockerfile para generar la imagen del JAR.

▪ docker-compose.yml orquesta todos los servicios: microservicios, bases de datos, config-server, gateway, frontend.

▪ Redes y volúmenes configurados para aislamiento y persistencia.

# 8. Jenkins CI/CD

▪ Jenkins ejecuta pipelines con las etapas: checkout, build (Maven), test (JUnit), empaquetado y despliegue.

▪ Jenkinsfile presente en cada microservicio para personalizar tareas.

▪ Logs de builds accesibles desde la interfaz de Jenkins.

# 9. Monitorización y Logs

▪ Spring Boot Actuator habilitado en cada microservicio para exponer métricas.

▪ Prometheus recolecta métricas desde los endpoints /actuator/prometheus.

▪ Grafana visualiza las métricas con dashboards personalizados.

▪ Logs enviados a Logstash y almacenados en Elasticsearch.

▪ Visualización de logs mediante Kibana.

# 10. Notificaciones y Otros Servicios

▪ api-citas y api-usuarios pueden enviar emails usando Spring Boot Mail.

▪ Configuración SMTP con Mailtrap o Gmail.

▪ Subida de imágenes desde frontend al backend, almacenamiento local inicialmente.

▪ Opción de persistencia en la nube con Amazon S3 o Cloudinary.

▪ Control de versiones REST mediante prefijos /api/v1/...