# Plan Detallado Paso a Paso - Proyecto Microservicios Cuidado del Bebé

📅 Inicio: 29/07/2025 | Fin estimado: 31/07/2025

* 🔸 Instala Java 17, Maven, Git y configura tu entorno (IDE recomendado: IntelliJ o Eclipse).
* 🔸 Crea un repositorio Git y carpetas para: config-server, api-usuarios, api-cuidados, etc.
* 🔸 Crea el microservicio 'config-server' con la dependencia 'spring-cloud-config-server'.
* 🔸 En la clase principal, añade la anotación @EnableConfigServer.
* 🔸 Crea un repositorio de configuración externo con archivos application.yml por microservicio.
* 🔸 Configura application.yml de config-server para apuntar al repositorio externo.
* 🔸 Verifica acceso: http://localhost:8888/api-usuarios/default (debe mostrar el YAML correspondiente).

📅 Inicio: 01/08/2025 | Fin estimado: 04/08/2025

* 🔸 Crea proyecto Spring Boot 'api-usuarios' con Web, Security, JPA, MySQL, Lombok, JWT y Swagger.
* 🔸 Crea entidad Usuario con campos: id, nombre, email, contraseña (encriptada), rol.
* 🔸 Crea DTOs de registro y login, y un repositorio con JpaRepository.
* 🔸 Implementa lógica JWT para generar y validar tokens.
* 🔸 Crea controlador REST con /auth/register y /auth/login.
* 🔸 Configura seguridad con Spring Security para proteger rutas.
* 🔸 Documenta todo con Swagger.

## Fase 3: Microservicio api-cuidados

📅 Inicio: 05/08/2025 | Fin estimado: 09/08/2025

* 🔸 Crea microservicio 'api-cuidados'.
* 🔸 Crea entidades: Biberon, Pañal, Suenio, vinculadas al usuario.
* 🔸 Crea repositorios, servicios y controladores REST con CRUD.
* 🔸 Configura conexión a MySQL y uso de Config Server.
* 🔸 Añade seguridad con JWT en todos los endpoints.
* 🔸 Documenta la API con Swagger.

## Fase 4: Frontend React - módulo cuidados

📅 Inicio: 10/08/2025 | Fin estimado: 16/08/2025

* 🔸 Crea app con Create React App. Instala: Material UI, Axios, React Router, JWT-decode.
* 🔸 Crea login y registro, guarda el token JWT en localStorage.
* 🔸 Diseña pantallas de biberones, pañales y sueño con Material UI.
* 🔸 Conecta a la API usando Axios con token JWT en headers.
* 🔸 Muestra errores con Snackbar y gestiona rutas protegidas.

## Fase 5: Microservicio api-gastos + Docker

📅 Inicio: 17/08/2025 | Fin estimado: 21/08/2025

* 🔸 Crea microservicio 'api-gastos'. Entidad: Gasto (usuarioId, categoría, cantidad, fecha, notas).
* 🔸 Implementa endpoints CRUD y filtros por mes o categoría.
* 🔸 Protege con JWT y documenta con Swagger.
* 🔸 Crea Dockerfile para api-gastos y docker-compose inicial con MySQL, config-server y api-gastos.
* 🔸 Verifica que api-gastos funcione dentro de Docker.

## Fase 6: Microservicio api-hitos + Jenkins básico

📅 Inicio: 22/08/2025 | Fin estimado: 25/08/2025

* 🔸 Crea microservicio 'api-hitos' con entidad Hito (usuarioId, título, fecha, descripción).
* 🔸 Endpoints CRUD + seguridad JWT + Swagger.
* 🔸 Crea Jenkinsfile simple en el proyecto.
* 🔸 Levanta Jenkins en local o Docker y configura un Job básico con build y test de api-hitos.

## Fase 7: Microservicio api-citas

📅 Inicio: 26/08/2025 | Fin estimado: 28/08/2025

* 🔸 Crea microservicio 'api-citas'. Entidad: Cita (fecha, motivo, tipo, usuarioId).
* 🔸 Endpoints CRUD, seguridad JWT, Swagger.
* 🔸 Filtra citas por fecha y tipo (vacuna, revisión, etc.).

## Fase 8: Microservicio api-diario

📅 Inicio: 29/08/2025 | Fin estimado: 31/08/2025

* 🔸 Crea 'api-diario'. Entidad: EntradaDiario (fecha, texto, usuarioId, emociones).
* 🔸 Endpoints para crear, leer, editar y eliminar entradas.
* 🔸 Protege con JWT y documenta con Swagger.

## Fase 9: Microservicio api-rutinas

📅 Inicio: 01/09/2025 | Fin estimado: 03/09/2025

* 🔸 Crea 'api-rutinas'. Entidad: Rutina (usuarioId, díaSemana, hora, tipoActividad).
* 🔸 CRUD básico, filtros por día de la semana.
* 🔸 Seguridad JWT + Swagger.

📅 Inicio: 04/09/2025 | Fin estimado: 08/09/2025

* 🔸 Completa configuración de 'api-gateway' con rutas a todos los microservicios.
* 🔸 Aplica filtros de seguridad JWT globalmente.
* 🔸 Verifica acceso desde el frontend a través del gateway.
* 🔸 Actualiza docker-compose con todos los servicios.
* 🔸 Configura Jenkins para compilar y testear todos los microservicios desde Git.

📅 Inicio: 29/08/2025 | Fin estimado: 07/09/2025

* 🔸 Crear estructura modular de carpetas para los componentes del frontend.
* 🔸 Implementar sistema de rutas con React Router para cada microservicio: cuidados, gastos, hitos, citas, diario y rutinas.
* 🔸 Configurar redirección automática si no hay token JWT.
* 🔸 Diseñar componente 'Navbar' con Material UI para navegación general.
* 🔸 Crear vistas de usuario: login, registro, perfil.
* 🔸 Crear vistas para gastos con formulario y tabla de resumen mensual.
* 🔸 Crear vista de hitos del bebé con tarjetas (cards) por evento.
* 🔸 Diseñar panel para gestión de citas con calendario.
* 🔸 Implementar editor simple para el diario personal del bebé.
* 🔸 Diseñar un panel diario de rutinas del bebé por día de la semana.

## Puertos de Microservicios y Servicios Asociados

|  |  |
| --- | --- |
| Microservicio / Servicio | Puerto |
| api-usuarios | 8080 |
| api-cuidados | 8081 |
| api-gastos | 8082 |
| api-hitos | 8083 |
| api-citas | 8084 |
| api-diario | 8085 |
| api-rutinas | 8086 |
| config-server | 8888 |
| api-gateway | 8090 |
| frontend React | 3000 |
| Jenkins | 8088 |
| api-bff | 8087 |

📅 Inicio: 08/09/2025 | Fin estimado: 17/09/2025

* 🔸 Configura una base de datos diferente para cada microservicio en docker-compose.yml.
* 🔸 Integra un servicio SMTP (como Mailtrap o Gmail SMTP) para enviar correos desde api-citas o api-usuarios.
* 🔸 Implementa subida de imágenes en api-diario y api-hitos, primero almacenando en disco local.
* 🔸 Explora opciones de almacenamiento gratuito en la nube como Amazon S3 (plan gratuito) o Cloudinary.
* 🔸 Agrega tests unitarios con JUnit en todos los servicios para modelos, controladores y servicios.
* 🔸 Implementa pruebas de integración con Spring Boot Test para validación de flujos reales.
* 🔸 Crea un sistema básico de notificaciones: email para citas o cambios importantes.
* 🔸 Prepara configuración para entornos locales y de producción usando Spring profiles.
* 🔸 Actualiza el docker-compose para incluir almacenamiento de imágenes y correo.
* 🔸 Revisa seguridad, validaciones y documenta endpoints finales en Swagger.

# Resumen de Microservicios y Funcionalidad

## api-bff (8087)

* 🔹 Servicio Backend for Frontend que orquesta llamadas a múltiples microservicios para unificar la información del Dashboard General de Bienvenida.
* 🔹 Endpoint único /api/v1/dashboard que devuelve rutinas del día, próximas citas, últimos cuidados y estadísticas rápidas del bebé.
* 🔹 Reduce la latencia y simplifica las llamadas desde el frontend, asegurando consistencia en la información mostrada.

## api-usuarios (8080)

* 🔹 Gestión de usuarios y autenticación.
* 🔹 Registro y login de usuarios.
* 🔹 Generación y validación de tokens JWT.
* 🔹 Protección de rutas según el token.

## api-cuidados (8081)

* 🔹 Registro de tomas, pañales, sueño y baño.
* 🔹 Historial por usuario y fecha.
* 🔹 CRUD completo de cada tipo de cuidado.

## api-gastos (8082)

* 🔹 Gestión de gastos del bebé.
* 🔹 Filtros por categoría y mes.
* 🔹 Cálculo de totales mensuales/anuales.

## api-hitos (8083)

* 🔹 Registro de hitos del desarrollo (primer paso, palabra, etc.).
* 🔹 Asociación de imágenes opcional.
* 🔹 Visualización cronológica.

## api-citas (8084)

* 🔹 Gestión de citas médicas y vacunas.
* 🔹 Notificaciones por email.
* 🔹 Visualización por tipo y fecha.

## api-diario (8085)

* 🔹 Diario personal del bebé y de la paternidad.
* 🔹 Entradas por día con texto e imágenes.
* 🔹 Organización cronológica.

## api-rutinas (8086)

* 🔹 Planificación diaria del bebé.
* 🔹 Rutinas por día de la semana.
* 🔹 Visualización organizada del plan diario.

## config-server (8888)

* 🔹 Servidor central de configuración.
* 🔹 Distribuye archivos application.yml.
* 🔹 Permite modificar la configuración sin reiniciar servicios.

## api-gateway (8090)

* 🔹 Punto de entrada único a todos los microservicios.
* 🔹 Gestión del enrutamiento.
* 🔹 Filtrado de tokens JWT a nivel global.
* 🔹 Oculta los puertos internos.

## Frontend React (3000)

* 🔹 Interfaz de usuario completa.
* 🔹 Pantallas para cada microservicio.
* 🔹 Enrutado con React Router.
* 🔹 Autenticación con JWT y Material UI para diseño.

## Jenkins (8088)

* 🔹 Automatización CI/CD.
* 🔹 Build, test y despliegue automático.
* 🔹 Pipeline para cada microservicio.

📅 Inicio: 18/09/2025 | Fin estimado: 24/09/2025

* 🔹 Implementar control de versiones en los endpoints REST (por ejemplo: /api/v1/usuarios).
* 🔹 Centralizar el manejo de errores usando la anotación @ControllerAdvice y clases de excepción personalizadas.
* 🔹 Añadir Spring Boot Actuator a cada microservicio para exponer métricas y endpoints de salud.
* 🔹 Integrar un stack de monitoreo con Prometheus y Grafana para visualizar rendimiento de los microservicios.
* 🔹 Configurar un sistema de logs centralizado usando ELK Stack (Elasticsearch, Logstash, Kibana) o alternativa más ligera (como Logtail o Papertrail).
* 🔹 Configurar alerta por logs de error críticos o servicios caídos (opcional, solo si se escala).
* 🔹 Actualizar documentación Swagger incluyendo la versión de cada API.

📅 Inicio: 18/09/2025 | Fin estimado: 24/09/2025

### Control de versiones en la API

🟡 ¿Por qué?: Permite mantener compatibilidad con futuras versiones de la API sin afectar a los clientes actuales.

🔧 ¿Cómo hacerlo?:

* ▪ En cada controlador REST, añade la versión en el path de la URL. Por ejemplo: @RequestMapping("/api/v1/usuarios").
* ▪ Si cambias la lógica o los datos de retorno, puedes crear un nuevo paquete con v2 y mantener ambos en paralelo.

### Gestión de errores centralizada con @ControllerAdvice

🟡 ¿Por qué?: Evita repetir el mismo código de manejo de errores en múltiples controladores y ofrece respuestas más claras al cliente.

🔧 ¿Cómo hacerlo?:

* ▪ Crea una clase con la anotación @ControllerAdvice.
* ▪ Define métodos con @ExceptionHandler para capturar excepciones específicas (como ResourceNotFoundException, etc).
* ▪ Devuelve objetos ResponseEntity con mensajes personalizados y códigos HTTP apropiados (404, 400, 500…).

### Monitorización con Spring Boot Actuator

🟡 ¿Por qué?: Te permite conocer el estado interno de tus microservicios en tiempo real (memoria, salud, beans, endpoints, etc).

🔧 ¿Cómo hacerlo?:

* ▪ Agrega la dependencia 'spring-boot-starter-actuator' al pom.xml de cada microservicio.
* ▪ Configura en application.yml qué endpoints quieres exponer, por ejemplo: management.endpoints.web.exposure.include=\*
* ▪ Accede a http://localhost:PORT/actuator/health para comprobar si el servicio está funcionando.

### Logs centralizados con ELK (Elasticsearch, Logstash, Kibana)

🟡 ¿Por qué?: En entornos con varios microservicios es difícil rastrear errores si cada uno guarda sus logs localmente. ELK permite ver todo en un solo lugar.

🔧 ¿Cómo hacerlo?:

* ▪ Usa Logstash como receptor de logs y configura Spring Boot para enviarle los logs (por TCP o Filebeat).
* ▪ Elasticsearch almacena los logs y Kibana los visualiza con filtros y gráficos.
* ▪ Alternativa más simple: servicios cloud como Logtail o Papertrail si no quieres levantar ELK en local.

### Stack de monitorización con Prometheus + Grafana

🟡 ¿Por qué?: Ideal para analizar el rendimiento de los microservicios, como tiempo de respuesta o uso de memoria.

🔧 ¿Cómo hacerlo?:

* ▪ Actuator expone métricas en /actuator/prometheus.
* ▪ Configura Prometheus para que recoja esas métricas de cada microservicio.
* ▪ Visualiza en Grafana con paneles personalizados.

### Alertas por errores o caídas

🟡 ¿Por qué?: En producción es vital que sepas rápidamente si algo falla, para actuar antes de que el usuario lo note.

🔧 ¿Cómo hacerlo?:

* ▪ Usa reglas de alerta en Grafana o tu sistema de logs (ELK, Logtail…).
* ▪ Ejemplo: 'si el número de errores 500 supera 10 en 5 minutos, enviar email o notificación'.

### Actualizar Swagger con versiones

🟡 ¿Por qué?: Ayuda a los desarrolladores a saber qué versión están utilizando y qué endpoints están disponibles.

🔧 ¿Cómo hacerlo?:

* ▪ En la configuración de Swagger, usa info().version("v1") dentro del bean Docket.
* ▪ Separa los endpoints de v1 y v2 en documentación distinta si usas controladores diferentes por versión.

## Fase 1: Inicialización del Proyecto y Repositorio

📅 Fechas estimadas: 01/08/2025 - 02/08/2025

### Crear repositorio Git para todo el proyecto

🟡 ¿Por qué?: Un control de versiones te permite mantener el historial de cambios, colaborar y revertir errores fácilmente.

🔧 ¿Cómo hacerlo?:

* ▪ Crea un repositorio en GitHub o GitLab con nombre identificativo.
* ▪ Clona el repositorio localmente: git clone <url>
* ▪ Inicializa tu estructura de carpetas con carpetas para backend, frontend, documentación, etc.
* ▪ Haz tu primer commit con un README.md básico.

### Definir estructura de microservicios

🟡 ¿Por qué?: Una estructura clara facilita el desarrollo modular, mantenimiento y escalabilidad.

🔧 ¿Cómo hacerlo?:

* ▪ Crea una carpeta por cada microservicio: api-usuarios, api-cuidados, etc.
* ▪ Cada microservicio debe tener su propio proyecto Spring Boot con pom.xml independiente.
* ▪ Prepara también carpetas para config-server y api-gateway desde el inicio.

## Fase 2: Creación de microservicios backend básicos

📅 Fechas estimadas: 03/08/2025 - 06/08/2025

### Crear microservicios con Spring Boot

🟡 ¿Por qué?: Spring Boot permite crear APIs REST rápidamente con una configuración mínima y fácil integración de dependencias.

🔧 ¿Cómo hacerlo?:

* ▪ Usa Spring Initializr (https://start.spring.io) para generar cada proyecto con: Spring Web, Spring Data JPA, MySQL Driver.
* ▪ Define el puerto y nombre de la aplicación en application.yml.
* ▪ Configura la conexión a la base de datos para cada microservicio.

### Definir modelos y controladores CRUD

🟡 ¿Por qué?: Los modelos representan los datos y los controladores exponen las operaciones para que el frontend pueda usarlas.

🔧 ¿Cómo hacerlo?:

* ▪ Crea entidades con anotaciones JPA (@Entity, @Id, @GeneratedValue).
* ▪ Crea repositorios que extiendan JpaRepository.
* ▪ Define controladores con @RestController y mapea rutas CRUD usando @GetMapping, @PostMapping, etc.

### Documentar APIs con Swagger

🟡 ¿Por qué?: Swagger facilita entender y probar los endpoints expuestos desde el navegador, sin necesidad de Postman.

🔧 ¿Cómo hacerlo?:

* ▪ Añade dependencia springdoc-openapi-ui en pom.xml.
* ▪ Accede a la documentación en /swagger-ui.html.
* ▪ Anota tus controladores con @Operation(summary = "...") para mejorar la legibilidad.

## Fase 3: Spring Security y Autenticación con JWT

📅 Fechas estimadas: 07/08/2025 - 09/08/2025

### Implementar Spring Security en api-usuarios

🟡 ¿Por qué?: Para proteger rutas sensibles y controlar el acceso a los recursos según si el usuario está autenticado o no.

🔧 ¿Cómo hacerlo?:

* ▪ Agrega dependencia de Spring Security en el pom.xml.
* ▪ Crea una clase de configuración que extienda WebSecurityConfigurerAdapter (o SecurityFilterChain si usas Spring Boot 3).
* ▪ Define rutas públicas (por ejemplo /auth/login, /auth/register) y rutas privadas (como /usuarios).

### Generar y validar tokens JWT

🟡 ¿Por qué?: JWT permite identificar al usuario sin mantener una sesión en el servidor. Cada petición incluye un token firmado que valida la identidad.

🔧 ¿Cómo hacerlo?:

* ▪ Agrega librería de JWT como 'jjwt'.
* ▪ Cuando un usuario se loguea correctamente, genera un token firmado con clave secreta y fecha de expiración.
* ▪ Crea un filtro que intercepte peticiones, extraiga el token y lo valide para permitir el acceso a rutas protegidas.

## Fase 4: Frontend React - módulo cuidados

📅 Fechas estimadas: 10/08/2025 - 16/08/2025

### Inicializar el proyecto React

🟡 ¿Por qué?: Para crear la interfaz gráfica donde interactuarán los usuarios con la aplicación.

🔧 ¿Cómo hacerlo?:

* ▪ Ejecuta: npx create-react-app bebe-cuidados
* ▪ Configura rutas básicas con React Router (por ejemplo: /login, /registro, /cuidados).
* ▪ Instala dependencias necesarias: Material UI, Axios, JWT-decode, React Router DOM.

### Crear pantallas de login, registro y cuidados

🟡 ¿Por qué?: Permiten a los usuarios autenticarse y registrar los datos del bebé (como biberón, pañal, sueño, etc).

🔧 ¿Cómo hacerlo?:

* ▪ Crea componentes separados para cada pantalla.
* ▪ Usa Material UI para crear formularios elegantes y accesibles.
* ▪ En el login, al autenticar, guarda el token en localStorage y úsalo en cada petición futura.

### Conectar con el backend con Axios

🟡 ¿Por qué?: Para que el frontend pueda guardar y recuperar datos desde las APIs.

🔧 ¿Cómo hacerlo?:

* ▪ Crea una instancia de Axios con cabecera Authorization con el JWT.
* ▪ Realiza peticiones GET, POST, PUT y DELETE a los endpoints de cuidados.
* ▪ Gestiona errores de red y muestra mensajes con Snackbar.

## Fase 5: Base de datos y Docker

📅 Fechas estimadas: 17/08/2025 - 18/08/2025

### Crear base de datos para cada microservicio

🟡 ¿Por qué?: Aísla cada dominio de la aplicación, evitando dependencias innecesarias y mejorando el escalado.

🔧 ¿Cómo hacerlo?:

* ▪ Define diferentes esquemas MySQL para cada microservicio (api\_gastos, api\_cuidados, etc).
* ▪ Configura application.yml con la URL, usuario y contraseña correspondientes en cada uno.
* ▪ Asegúrate de usar nombres únicos para evitar conflictos.

### Contenerizar microservicios con Docker

🟡 ¿Por qué?: Permite levantar todos los servicios de forma automática y replicable en cualquier entorno.

🔧 ¿Cómo hacerlo?:

* ▪ Crea un Dockerfile por microservicio que compile el JAR con Maven y lo ejecute.
* ▪ Define un docker-compose.yml que contenga: servicios para MySQL, microservicios, gateway, config-server.
* ▪ Incluye volúmenes para persistir la base de datos.

## Fase 6: Implementar Config Server y Eureka (opcional)

📅 Fechas estimadas: 19/08/2025 - 20/08/2025

### Usar Spring Cloud Config Server

🟡 ¿Por qué?: Permite mantener la configuración externa (application.yml) de todos los servicios en un solo lugar y actualizarlas fácilmente.

🔧 ¿Cómo hacerlo?:

* ▪ Crea un microservicio nuevo con Spring Boot llamado config-server.
* ▪ Agrega dependencia spring-cloud-config-server.
* ▪ Apunta cada microservicio al config-server en su bootstrap.yml.

### Registrar servicios en Eureka (opcional)

🟡 ¿Por qué?: Centraliza el descubrimiento de servicios, útil cuando tienes múltiples instancias balanceadas.

🔧 ¿Cómo hacerlo?:

* ▪ Agrega dependencia spring-cloud-starter-netflix-eureka-server en un microservicio nuevo.
* ▪ Marca cada microservicio con @EnableEurekaClient.
* ▪ Apunta gateway y microservicios al servidor de Eureka.

## Fase 7: Microservicios adicionales: gastos, hitos, diario, citas, rutinas

📅 Fechas estimadas: 21/08/2025 - 25/08/2025

### Desarrollar lógica específica para cada nuevo microservicio

🟡 ¿Por qué?: Cada módulo del backend cubre funcionalidades independientes que deben estar separadas para mantener bajo acoplamiento.

🔧 ¿Cómo hacerlo?:

* ▪ Crea las entidades correspondientes: Gasto, Hito, EntradaDiario, Cita, Rutina.
* ▪ Diseña sus controladores CRUD y servicios usando buenas prácticas.
* ▪ Agrega validaciones con anotaciones como @NotNull y @Size.
* ▪ Prepara datos de ejemplo para pruebas con @DataLoader o import.sql.

## Fase 8: Integración completa desde el frontend

📅 Fechas estimadas: 26/08/2025 - 30/08/2025

### Conectar todos los microservicios desde React

🟡 ¿Por qué?: El frontend debe comunicarse correctamente con cada API para ofrecer funcionalidades reales al usuario.

🔧 ¿Cómo hacerlo?:

* ▪ Crea servicios de frontend que usen Axios para interactuar con cada API.
* ▪ Agrega formularios con validaciones para cada entidad (gasto, cita, rutina, etc).
* ▪ Muestra listados con Material UI DataGrid u otros componentes visuales.
* ▪ Maneja estados de carga y error con React Context o useReducer.

## Fase 9: Autenticación completa y control de acceso en frontend

📅 Fechas estimadas: 31/08/2025 - 01/09/2025

### Proteger rutas en React con rutas privadas

🟡 ¿Por qué?: Para evitar que usuarios no logueados accedan a datos sensibles del bebé.

🔧 ¿Cómo hacerlo?:

* ▪ Crea un componente de ruta privada que verifique la existencia del JWT en localStorage.
* ▪ Redirige automáticamente a /login si el usuario no está autenticado.
* ▪ En cada petición, añade el JWT a la cabecera Authorization.

## Fase 10: Despliegue local completo con Docker

📅 Fechas estimadas: 02/09/2025 - 04/09/2025

### Levantar todos los servicios con docker-compose

🟡 ¿Por qué?: Facilita el entorno de desarrollo y pruebas integradas con un solo comando.

🔧 ¿Cómo hacerlo?:

* ▪ Define en docker-compose todos los contenedores necesarios: microservicios, MySQL, frontend, gateway, config-server.
* ▪ Asocia cada microservicio a su base de datos usando variables de entorno.
* ▪ Incluye red propia para que los servicios se comuniquen entre sí.
* ▪ Levanta con docker-compose up --build y comprueba logs.

## Fase 11: Integración con Jenkins CI/CD

📅 Fechas estimadas: 05/09/2025 - 07/09/2025

### Crear pipeline básico de integración continua

🟡 ¿Por qué?: Automatiza la compilación, pruebas y despliegue, reduciendo errores manuales y acelerando el ciclo de vida del desarrollo.

🔧 ¿Cómo hacerlo?:

* ▪ Instala Jenkins en local (en contenedor o máquina).
* ▪ Crea un pipeline con etapas: checkout de código, build con Maven, test con JUnit, empaquetado del JAR.
* ▪ Agrega notificaciones o logs para ver el resultado de cada ejecución.
* ▪ Ejecuta el despliegue en contenedor automáticamente si los tests pasan.

## Fase 12: Funcionalidades Avanzadas y Producción

📅 Fechas estimadas: 08/09/2025 - 17/09/2025

### Subida de imágenes y persistencia en la nube

🟡 ¿Por qué?: Permite guardar recuerdos (hitos, entradas del diario) de forma permanente y segura.

🔧 ¿Cómo hacerlo?:

* ▪ Permite subir imágenes desde React con un campo tipo file.
* ▪ Recibe el archivo en el backend y guárdalo primero en disco local.
* ▪ Integra Amazon S3 o Cloudinary para almacenamiento en la nube y devuelve URL accesible.
* ▪ Guarda la URL en la base de datos.

### Servicio SMTP para emails de notificación

🟡 ¿Por qué?: Mejora la experiencia del usuario recordándole citas o eventos importantes automáticamente.

🔧 ¿Cómo hacerlo?:

* ▪ Usa Spring Boot Mail (starter-mail) en el backend.
* ▪ Configura SMTP (por ejemplo, Mailtrap o Gmail).
* ▪ Envía emails desde eventos como nueva cita o recordatorio diario.

### Tests unitarios y de integración

🟡 ¿Por qué?: Garantiza que las funcionalidades están correctamente implementadas y que no se rompen al añadir nuevas.

🔧 ¿Cómo hacerlo?:

* ▪ Escribe tests unitarios para cada servicio con JUnit y Mockito.
* ▪ Implementa pruebas de integración que simulan casos reales de uso con Spring Boot Test.
* ▪ Ejecuta los tests en el pipeline de Jenkins para validación automática.

### Notificaciones por email (y opcionalmente push)

🟡 ¿Por qué?: Permite a los padres estar al tanto de los eventos importantes incluso si no están dentro de la app.

🔧 ¿Cómo hacerlo?:

* ▪ Envía notificaciones por email con asunto y cuerpo personalizados.
* ▪ Opcionalmente, estudia el uso de Firebase Cloud Messaging (FCM) para notificaciones push si quieres extender la app móvil.

## Fase 13: Microservicio api-bff (Backend For Frontend - Orquestador de Dashboard)

📅 Fechas estimadas: 18/09/2025 - 22/09/2025  
  
Objetivo:  
Crear un microservicio Backend for Frontend (BFF) que unifique y orqueste llamadas a múltiples microservicios  
(api-cuidados, api-citas, api-rutinas, profile-service) para proporcionar al frontend un único endpoint que  
devuelva toda la información necesaria para el Dashboard General de Bienvenida.  
  
🟡 ¿Por qué?:   
Actualmente el frontend necesita llamar a varios servicios para obtener la información del dashboard.   
El BFF optimiza el rendimiento, reduce latencia y centraliza la lógica de agregación de datos.  
  
🔧 ¿Cómo hacerlo?:  
  
▪ Crear un nuevo proyecto Spring Boot llamado 'api-bff' con dependencias: Web, Security, OpenFeign, Validation, Lombok, JWT, Swagger.  
▪ Implementar seguridad con Spring Security y validación de JWT, reenviando el token a los microservicios internos.  
▪ Configurar clientes Feign para: rutinas-service, citas-service, cuidados-service y profile-service.  
▪ Endpoint principal: GET /api/v1/dashboard, que recibe opcionalmente bebeId (si no se pasa, se obtiene el perfil activo del usuario).  
▪ Orquestar las llamadas en paralelo usando ExecutorService (Java 8) para obtener:  
 - Rutinas del día (rutinas-service).  
 - Próximas citas (citas-service).  
 - Últimos cuidados y estadísticas rápidas (cuidados-service).  
▪ Implementar tolerancia a fallos con Resilience4j (timeouts, circuit breakers, fallbacks).  
▪ Unificar datos en un DashboardDTO que incluya fecha, datos del bebé, rutinasHoy, proximasCitas, ultimosCuidados, estadisticasRapidas y estado de cada bloque.  
▪ Documentar API con Swagger y añadir ejemplos de respuesta.  
▪ Añadir métricas (Actuator + Prometheus) para monitorizar tiempos de respuesta y errores parciales.  
  
📦 Despliegue:  
▪ Crear Dockerfile y añadir api-bff al docker-compose con configuración de red y variables de entorno necesarias.  
▪ Incluir en el api-gateway para exponer el endpoint /dashboard a React.  
▪ Añadir al pipeline Jenkins para build, test y despliegue automático.  
  
✅ Resultado esperado:  
Un único endpoint optimizado que provea al frontend todos los datos necesarios para renderizar el Dashboard  
de Bienvenida de forma rápida y eficiente.