Prueba Técnica para Desarrollador Mobile - Aplicación Ionic

Today Task — Ionic + Angular (Cordova)

Aplicación móvil tipo To-Do / Task Manager construida con Ionic + Angular, orientada a funcionar como app híbrida para Android e iOS usando Cordova.

Esta aplicación permite:

- Crear, editar y eliminar tareas.
- Marcar tareas como completadas (con fecha de finalización).
- Organizar tareas en categorías personalizadas (crear, editar, eliminar).
- Asignar una categoría a cada tarea.
- Visualizar tareas por categoría.
- Habilitar/ocultar funcionalidades mediante feature flags con Firebase Remote Config.

1. Stack Tecnológico

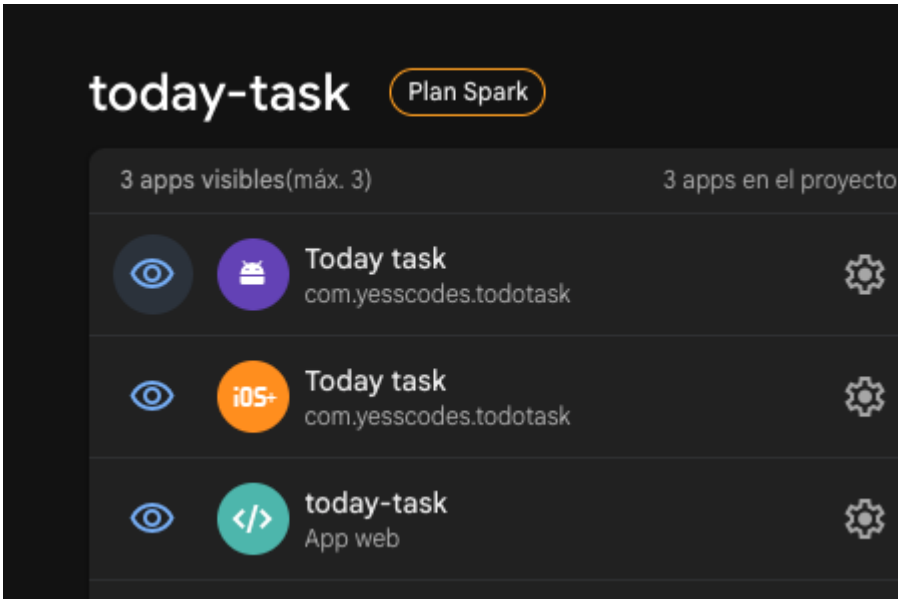
- Framework UI: Ionic + Angular (Standalone Components).
- Runtime nativo: Cordova.
- Lenguaje: TypeScript / HTML / SCSS.
- Almacenamiento local: localStorage (servicios CategoryStorageService y TaskStorageService).
- Integración en la nube: Firebase (Remote Config).
- Gestos móviles: createGesture de Ionic para swipe en tarjetas.
- Plataformas objetivo: Android & iOS.

2. Versionamiento de la aplicación

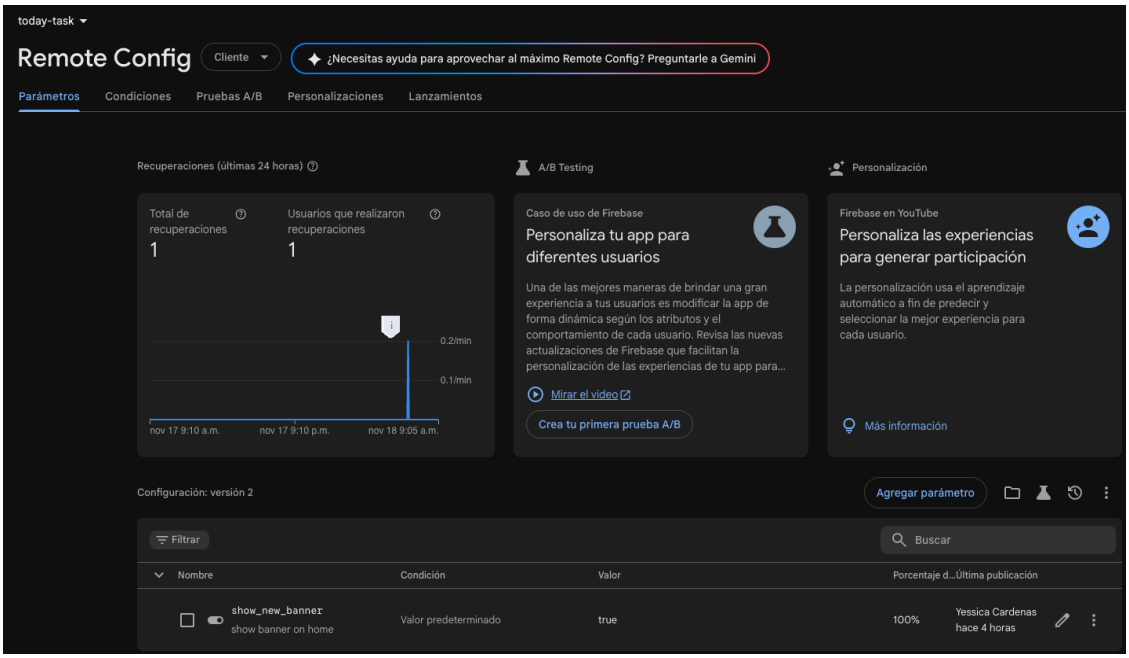
- Se crea proyecto en Github https://github.com/1Jessie9/today-task-base
- Al proyecto se sube un commit inicial con los componente iniciales
- Se crea un fork del proyecto base https://github.com/1Jessie9/today-task
- Y en el fork se crea una rama para trabajar la nueva funcionalidad de remote flag de Firebase https://github.com/1Jessie9/today-task/tree/feature/firebase

3. Implementación de Firebase y Remote Config

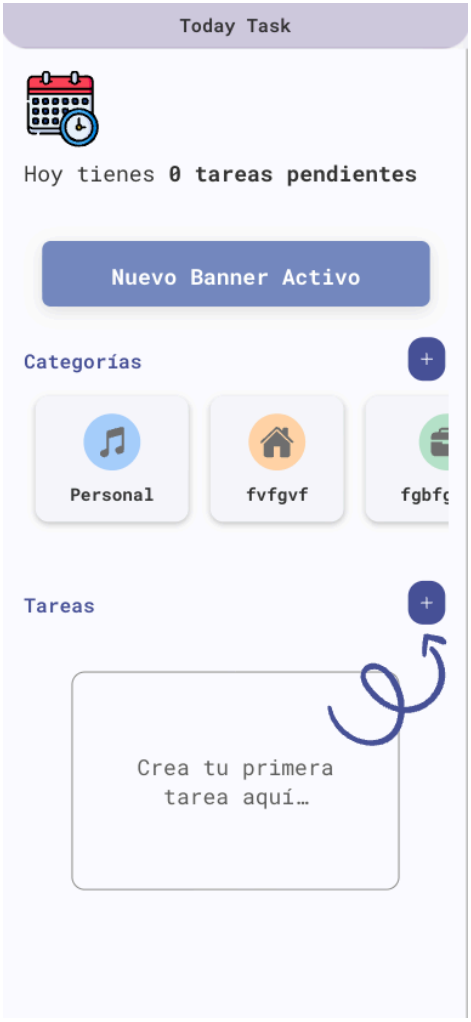
- Se crea aplicación en firebase y se configura plataformas de IOS y Android



- Se configura feature flag para que aparezca nueva funcionalidad de banner en el home



- Así mismo cuando se activa el flag se debe activar en la aplicación



4. Optimización de Rendimiento

5. Archivos APK y IPA

Se proporciona archivo APK Android: app_android_todotask

Se proporciona archivo IPA IOS: app_ios_todotask

6. Respuestas a las siguientes preguntas

o ¿Cuáles fueron los principales desafíos que enfrentaste al implementar las nuevas funcionalidades?

Fue desafiante para mí trabajar con cordova, estoy acostumbrada a trabajar con capacitor para Ionic, ya que tiene a estar más actualizado y estable, sin embargo aunque fue retante y logré culminar la mayoría de los retos que me propuse

o ¿Qué técnicas de optimización de rendimiento aplicaste y por qué?

Algunas técnicas / decisiones aplicadas:

- Almacenamiento local ligero:

Se usa localStorage con servicios dedicados (CategoryStorageService, TaskStorageService) para evitar llamadas de red y mantener acceso rápido a los datos.

- Filtrado de tareas completadas en servicio:

Las tareas completadas se marcan con completed: true y completedAt, pero se filtran en el servicio para no renderizar innecesariamente en la lista principal.

- Ordenamiento por prioridad:

Las tareas se ordenan en memoria por prioridad (high, medium, low) antes de ser devueltas a la UI, minimizando trabajo en el componente y manteniendo la UX clara.

- Componentes reutilizables:

    - CardTaskComponent encapsula lógica de gestos y swipe.
    - CategoryCarouselComponent evita duplicar código del swiper en home y create-task.

Esto reduce la lógica duplicada y hace que la app sea más fácil de mantener.

- Paginación / manejo de grandes volúmenes (estrategia preparada):

Aunque el demo usa un tamaño moderado de tareas, la estructura de servicios permite introducir filtros, paginación o “carga incremental” sin romper la UI.

- Se usa la extensión de Google Chrome **lighthouse** para obtener reportes del performance y la calidad

¿Cómo aseguramos la calidad y mantenibilidad del código?

1. Separación de responsabilidades:
    a. Servicios: lógica de negocio y acceso a datos.
    b. Páginas: orquestan la UI y llaman a los servicios.
    c. Componentes compartidos: elementos reutilizables.
    d. Pipes: transformaciones visuales.
2. Nombres claros y consistentes:
    a. Interfaces definidas para cada objeto o información.
    b. Claves de localStorage centralizadas en keys.ts.

3. Uso de componentes Standalone de Angular:
    a. Cada página y componente declara explícitamente sus dependencias (imports).
    b. Facilita el mantenimiento y el lazy loading si se quiere agregar más adelante.

4. Estructura preparada para testeo:

La lógica principal vive en servicios puros, lo que facilita escribir pruebas unitarias sobre creación, actualización, completado y borrado de tareas/categorías sin depender directamente de Ionic.