Respuestas Grupo 5

August 14, 2024

1 Seguimiento para el despliegue

1. ¿Cómo está estructurada nuestra aplicación? ¿Monorepo o repositorios separados?

   - Ok, se implementan repositorios separados para frontend y backend: Revisar y refinar estructura de directorios
   - **Observación/Acción:** Detallar la estructura de directorios para frontend y backend. Definir convenciones de nombrado y organización de archivos.

2. ¿Cómo manejaremos las variables de entorno en diferentes entornos?

   - Implementación de dotenv con variables básicas como puertos de servidor y cliente. Trabajo futuro: la inclusión de variables sensibles como api Keys, etc.
   - **Observación/Acción:** Implementar un sistema seguro para manejar claves API y credenciales en diferentes entornos.

3. ¿Necesitamos ajustar la configuración para producción vs desarrollo?

   - Trabajo Futuro: Redis, Definición de pipelines para integración continua, deshabilitar modos de debuging para producción, generar versiones minimificadas del código
   - **Observación/Acción:** Establecer configuraciones específicas para producción y desarrollo ahora. Implementar minimización de código y deshabilitación de modo debug para producción de inmediato.

4. ¿Hemos optimizado el rendimiento de nuestra aplicación React?

   - Implementar lazy loading, code spling,
   - **Observación/Acción:** Además de lazy loading y code splitting, implementar memoización con React.memo y useMemo para componentes y cálculos costosos. Optimizar re-renderizados con useCallback.

5. ¿Cómo manejaremos la carga de assets estáticos?
    - Mediante lazy loading para assets internos y uso de CNDs (o herramientas como TailwindCSS) para assets externos
    - **Observación/Acción:** Implementar compresión de imágenes y otros assets. Considerar el uso de formatos modernos como WebP para imágenes.
6. ¿Estamos utilizando code-splitting para mejorar los tiempos de carga?
    - No. Configurar el Bundler para code-splitting
    - **Corrección**: Sí, es posible implementar code-splitting. Se logra utilizando la función `import()` de JavaScript para importaciones dinámicas, junto con React.lazy() y Suspense para componentes. Webpack, el bundler más comúnmente usado con React, soporta code-splitting automáticamente cuando detecta estas importaciones dinámicas.
    - **Observación/Acción:** Implementar code-splitting en las rutas principales y componentes pesados de la aplicación.
7. ¿Cómo manejaremos las conexiones a las bases de datos en producción?
    - Mediante el controlador, usando los secretos contenidos en dotenv.
    - **Observación/Acción:** Detallar la estrategia de manejo de conexiones. Implementar un pool de conexiones con límites adecuados. Considerar el uso de ORMs para mayor abstracción y seguridad.
8. ¿Hemos implementado medidas de seguridad adecuadas (CORS, Helmet, etc.)?
    - CORS ya está falta definir los orígenes permitidos. Trabajo futuro implementar Helmet y otras librerías de seguridad
    - **Observación/Acción:** Completar la configuración de CORS, definiendo los orígenes permitidos. Implementar Helmet ahora para protección básica de headers HTTP.
9. ¿Cómo gestionaremos los logs en producción?
    - Deshabilitaremos los debuggers
    - **Observación/Acción:** Implementar un sistema de logging robusto (por ejemplo, Winston o Bunyan) que permita diferentes niveles de log y rotación de archivos. Considerar la integración con servicios de monitoreo de logs.
10. ¿Cómo realizaremos las migraciones de base de datos?
    - Trabajo Futuro: Implementación de Knex.js u otras tecnologías propias de la plataforma donde se despliegue la BD para permitir migraciones y backups

- **Observación/Acción:** Comenzar la implementación de un sistema de migraciones ahora. Si se usa un ORM, utilizar sus herramientas de migración integradas.

11. ¿Tenemos un plan de backup y recuperación?

    - Trabajo Futuro: Implementación de Knex.js u otras tecnologías propias de la plataforma donde se despliegue la BD para permitir migraciones y backups
    - **Observación/Acción:** Establecer frecuencia de backups y probar el proceso de recuperación.

12. ¿Cómo manejaremos las conexiones en un entorno de alta concurrencia?

    - Mediante un pool de conexiones con 10 conexiones concurrentes. Trabajo Futuro: Dimensionar el pool y definir estrategias de escalabilidad de BD
    - **Observación/Acción:** Investigar y determinar el tamaño óptimo del pool basado en las especificaciones del servidor y las necesidades de la aplicación. Implementar monitoreo para ajustar según sea necesario.

13. ¿Tenemos suficiente cobertura de pruebas unitarias y de integración?

    - No, faltan las pruebas de Fontend y de integración, probar la correctitud de las pruebas
    - **Observación/Acción:** Establecer un objetivo de cobertura de pruebas (por ejemplo, 80%) y comenzar a escribir pruebas para alcanzarlo. Priorizar pruebas para funcionalidades críticas.

14. ¿Cómo integraremos las pruebas en nuestro pipeline de CI/CD?

    - Automatizar 5 pruebas con Github actions
    - **Observación/Acción:** Ampliar el alcance de las pruebas automatizadas. Incluir pruebas unitarias, de integración y e2e en el pipeline. Configurar la ejecución de todas las pruebas en cada push.

15. ¿Hemos considerado pruebas de carga y estrés?

    - Trabajo Futuro: Pruebas de carga y estrés
    - **Observación/Acción:** Comenzar a planificar e implementar pruebas de carga básicas ahora. Utilizar herramientas como Apache JMeter o k6 para simular carga en la aplicación.

16. ¿Qué plataforma de despliegue es más adecuada para nuestras necesidades?

    - Railway.app

- **Observación/Acción:** Justificar la elección de Railway.app. Comparar con otras opciones (por ejemplo, Heroku, DigitalOcean) en términos de costos, escalabilidad y facilidad de uso.

17. ¿Cómo configuraremos nuestro pipeline de CI/CD?

    - **Observación/Acción:** Configurar un pipeline de CI/CD utilizando GitHub Actions. Este pipeline incluirá pasos para la instalación de dependencias, ejecución de pruebas automatizadas, construcción del proyecto, y despliegue automático a Railway.app cuando se hagan push a la rama principal. Imprlementar controles de calidad como linting y comprobación de tipos antes del despliegue.

18. ¿Tenemos un plan para el rollback en caso de problemas post-despliegue?

    - Trabajo Futuro: definir plan Rollback
    - ¿Qué herramientas utilizaremos para monitorear el rendimiento y los errores?
      - Implementar Lighthouse. Trabajo Futuro: Monitoreo de errores
      - **Observación/Acción:** Además de Lighthouse, implementar una solución de monitoreo de errores en producción (por ejemplo, Sentry). Considerar herramientas de APM como New Relic o Datadog para monitoreo completo.
    - ¿Cómo manejaremos las actualizaciones y parches de seguridad?
      - Mediante las ramas y las pipeline. Trabajo Futuro: Dependabot y npm Audit
      - **Observación/Acción:** Configurar Dependabot y npm audit ahora para recibir alertas de seguridad. Establecer un proceso regular de revisión y aplicación de parches.
    - ¿Tenemos un plan de escalabilidad para manejar aumentos de tráfico?
      - No. Trabajo futuro: llevar a cloud con el uso de balaceador de carga
      - **Observación/Acción:** Desarrollar un plan de escalabilidad básico ahora. Investigar opciones de auto-escalado en Railway.app o considerar la migración a una plataforma que ofrezca estas características si es necesario.