🧪 Trabajo Práctico – Dockerizando una App Fullstack (Front + Back)

Repositorio general: https://github.com/ZERO-JIFAR/arnold_metodolo
Repositorio del TP: https://github.com/ZERO-JIFAR/arnold_metodologia/tree/main/apiAndFront

🌐 Sobre NGINX y el frontend

1. ¿Por qué se usa NGINX para servir el frontend y no vite preview o npm run dev?
   Porque vite preview y npm run dev están pensados solo para desarrollo. En producción se necesita un servidor robusto, eficiente y seguro como NGINX, que:
   - Sirve archivos estáticos con mejor rendimiento.
   - No tiene dependencias de Node.js.
   - Tiene menor consumo de recursos.
   - Maneja configuraciones avanzadas como compresión, cacheo, redirecciones, etc.

2. ¿Qué beneficios tiene usar NGINX en producción (rendimiento, seguridad)?
   - Rendimiento: sirve archivos estáticos más rápido que Node.js o Vite.
   - Caching: permite cachear recursos (como JS/CSS) y mejorar tiempos de carga.
   - Seguridad: puede bloquear IPs, limitar peticiones, ocultar headers, etc.
   - Escalabilidad: es liviano, ideal para contenedores y balanceadores de carga.

3. ¿Qué diferencia hay entre usar nginx:alpine vs. nginx?
   - nginx:alpine usa la versión Alpine Linux, una distribución minimalista y liviana (~5MB), ideal para producción.
   - nginx (sin alpine) es más pesada (~133MB) porque incluye más herramientas y utilidades.

4. ¿Dónde está alojado el contenido que NGINX sirve dentro del contenedor?
   En: /usr/share/nginx/html

🧱 Sobre la construcción de imágenes

5. ¿Qué ventajas ofrece una imagen multietapa (builder + nginx)?
   - Tamaño reducido: se elimina Node y otras herramientas después del build.
   - Más segura: la imagen final solo contiene archivos estáticos y NGINX.
   - Separación de responsabilidades: una etapa se encarga de construir, otra de servir.

6. ¿Por qué copiamos package*.json antes de copiar el resto del código?
   Para aprovechar la cache de Docker.
   Si package.json no cambió, se reutiliza la capa con las dependencias instaladas y no se ejecuta npm install de nuevo.

🔗 Sobre la conexión entre servicios

7. ¿Por qué no se usa localhost para conectar servicios dentro de Docker?
   Porque dentro de Docker, cada contenedor tiene su propia red. localhost dentro de un contenedor apunta a sí mismo, no a otro servicio.
   Ejemplo: para que el frontend se conecte al backend, debe usar el nombre del servicio en docker-compose, como http://backend:3000.

8. ¿Qué beneficios aporta usar una red compartida en docker-compose?
    - Conectividad automática entre servicios sin configuración extra.
    - Resolución de nombres por DNS interna (backend, db, etc.).
    - Aislamiento del resto de la máquina: los contenedores solo se ven entre ellos.

🧠 Sobre buenas prácticas

9. ¿Por qué separar frontend y backend en imágenes distintas?
    - Despliegue independiente (actualizas solo el frontend si cambia).
    - Escalabilidad y mantenimiento más fácil.
    - Imágenes más pequeñas y específicas para cada rol.

10. ¿Qué problemas genera no usar .dockerignore?
    - Se copian archivos innecesarios (como .git, node_modules, tests, etc.).
    - Hace más lento el build.
    - Puede filtrar archivos sensibles (credenciales, claves, etc.).

11. ¿Por qué es importante exponer puertos en Dockerfile?
    - Sirve como documentación interna del contenedor (no abre el puerto automáticamente).
    - Permite que herramientas externas detecten qué puertos están pensados para usarse.

12. ¿En qué situaciones sería mejor usar docker run en lugar de docker-compose?
    - Pruebas rápidas o contenedores temporales.
    - Ejecutar una imagen sin necesidad de redes ni dependencias (por ejemplo, nginx como server estático).
    - Casos simples donde no necesitas múltiples servicios.

13. ¿Por qué es mejor usar variables de entorno que hardcodear URLs?
    - Flexibilidad: cambias configuraciones sin tocar el código.
    - Facilita despliegues en distintos entornos (dev, prod, staging).
    - Mejor manejo de secretos (como claves API, sin subirlas al repositorio).