



# PRUEBA TÉCNICA DESARROLLADOR FULLSTACK

## Proyecto: SyncSpace

Duración estimada:	5-7 días laborales
Nivel:	Semi-Senior / Senior
Stack principal:	Next.js 15 + Ruby on Rails + PostgreSQL

## DESCRIPCIÓN GENERAL

SyncSpace es una aplicación web moderna para compartir ideas, discutir temas y reaccionar en tiempo real. Combina una arquitectura modular con comunicación en vivo mediante WebSockets y una interfaz dinámica, fluida y responsive. El proyecto busca evaluar tus habilidades en desarrollo fullstack, arquitectura de software, optimización de rendimiento y experiencia de usuario.

## ESTRUCTURA DEL PROYECTO

El proyecto debe organizarse en un monorepo con la siguiente estructura:

- **client/** → Frontend desarrollado en Next.js 15 con TypeScript, Tailwind CSS y Axios
- **server/** → Backend API RESTful en Ruby on Rails con autenticación JWT y Docker
- **docs/** → Documentación técnica, diagramas de arquitectura y decisiones de diseño
- **README.md** → Instrucciones de instalación, ejecución y despliegue

# TECNOLOGÍAS REQUERIDAS

## Frontend

- Next.js 15 (versión más reciente) con App Router y TypeScript
- Tailwind CSS para estilos
- Axios para peticiones HTTP
- Socket.io o similar para comunicación en tiempo real con el backend
- next-intl o react-intl para internacionalización
- Jest y React Testing Library para testing
- **Deploy:** Vercel o Netlify

## Backend

- Ruby on Rails 7+ en modo API
- PostgreSQL como base de datos principal
- JWT para autenticación y autorización
- ActionCable o Socket.io para WebSockets
- RSpec para testing
- Redis para caché y gestión de sesiones (opcional pero recomendado)
- **Dockerización:** Backend completamente dockerizado con Docker Compose
- **Deploy:** Render.com o Fly.io

## Arquitectura de Despliegue

**Importante:** El frontend y backend se despliegan de forma independiente:

- **Frontend:** Deploy en Vercel o Netlify (sin Docker, usando sus plataformas nativas)
- **Backend:** Deploy en Render.com o Fly.io usando Docker
- Configuración de CORS para permitir comunicación entre ambos dominios
- Variables de entorno separadas para cada ambiente (desarrollo, producción)

# REQUERIMIENTOS FUNCIONALES

## 1. Autenticación y Seguridad

Implementar un sistema completo de autenticación con las siguientes características:

- **Registro de usuarios:** name, email, password, password\_confirmation, profile\_picture (opcional)
- **Login con JWT:** Tokens firmados y verificados en cada petición
- **Confirmación de email:** Envío de correo de verificación al registrarse
- **Recuperación de contraseña:** Sistema de reset mediante email
- **Middleware de autorización:** Validación de tokens en endpoints protegidos
- **CORS configurado:** Permitir peticiones desde el dominio del frontend desplegado
- **Refresh tokens:** Implementación de renovación automática de sesión

## 2. Gestión de Usuarios

- **Perfil de usuario:** Visualización y edición de nombre, foto, biografía
- **Perfil público:** Vista con todos los posts del usuario y estadísticas (total de posts, reacciones recibidas, comentarios recibidos)
- **Configuración de preferencias:** Tema (modo oscuro/claro) e idioma

## 3. Sistema de Posts

- **Campos del post:** title, description, picture (opcional), tags[], user\_id, timestamps
- **CRUD completo:** Solo el autor puede editar y eliminar sus posts
- **Visualización:** Mostrar contadores de reacciones y comentarios en tiempo real
- **Últimos 3 comentarios:** Previsualización de los comentarios más recientes
- **Scroll infinito:** Carga progresiva de 10 posts por página
- **Actualización en tiempo real:** Cuando el usuario está en la parte superior del feed, los nuevos posts aparecen automáticamente. Si está desplazado hacia abajo, mostrar un banner tipo '■ 3 nuevos posts disponibles' con opción de clic para cargarlos
- **Ordenamiento:** Posts más recientes primero

## 4. Buscador Avanzado

- Búsqueda por **título del post**
- Búsqueda por **nombre de usuario** (autor)
- Búsqueda por **etiqueta/tag**

- Búsqueda combinada con sintaxis especial (ej: '@daniel tag:tech título:ruby')
- Resultados actualizados dinámicamente mientras se escribe
- Optimización con índices en PostgreSQL para búsquedas rápidas
- Destacar términos coincidentes en los resultados

## 5. Sistema de Etiquetas

- Posts con **múltiples etiquetas** (relación many-to-many)
- Vista dedicada de '**Posts por etiqueta**'
- Autocompletado de etiquetas en el formulario de creación
- Asignación de colores personalizados para cada etiqueta
- Creación automática de etiquetas si no existen

## 6. Sistema de Comentarios

- **Campos:** description, user\_id, post\_id, parent\_comment\_id (para respuestas anidadas)
- **Comentarios anidados:** Permitir respuestas a comentarios (un nivel de anidación)
- **CRUD:** Los usuarios solo pueden editar/eliminar sus propios comentarios
- **Ordenamiento:** Comentarios más recientes primero
- **Actualización en tiempo real:** Nuevos comentarios aparecen instantáneamente vía WebSocket
- **Validaciones:** Longitud mínima y máxima, prevención de contenido vacío

## 7. Sistema de Reacciones

- **Tipos de reacciones:** 'like' (■), 'love' (♥■), 'dislike' (■)
- Aplicable tanto a **posts** como a **comentarios**
- Un usuario puede dar **una sola reacción por tipo**
- Clic repetido en la misma reacción la **elimina** (toggle)
- Contadores actualizados instantáneamente en toda la aplicación
- Visualización de quiénes reaccionaron (opcional, mostrar al hacer hover)

## 8. Notificaciones en Tiempo Real

Sistema completo de notificaciones que se generan automáticamente cuando:

- Alguien comenta en tus posts
- Responden a tus comentarios
- Te mencionan usando @usuario o @email
- Reaccionan a tu contenido (posts o comentarios)
- **Actualización instantánea** mediante WebSocket
- **Icono con badge** mostrando el número de notificaciones no leídas
- Página dedicada **/notifications** para ver el historial completo
- Marcar notificaciones como leídas

- Preparado para integración futura con notificaciones push del navegador

## 9. Tema y Personalización

- **Modo oscuro/claro:** Toggle persistente almacenado en localStorage
- **Sincronización con el sistema:** Detectar preferencias del sistema operativo
- **Internacionalización:** Soporte para Español e Inglés
- **Detección automática:** Idioma del navegador como predeterminado
- **Selector manual:** Switch en el menú de usuario para cambiar idioma

## 10. Página Principal (Index)

### Usuario no autenticado:

- Visualiza posts más populares (ordenados por reacciones + comentarios)
- No puede crear posts, comentar ni reaccionar
- Botón prominente de 'Registrarse' o 'Iniciar sesión'

### Usuario autenticado:

- Feed principal con posts más recientes
- Cada post muestra: número de reacciones, número de comentarios, últimos 3 comentarios
- Scroll infinito con carga incremental
- Inserción automática de nuevos posts en la parte superior
- Banner de 'nuevos posts disponibles' cuando hay actualizaciones fuera del viewport

## 11. Sección 'Mis Posts'

- Página dedicada al contenido del usuario autenticado
- Listado de todos sus posts con opciones de editar/eliminar
- **Estadísticas personales:**
  - - Total de posts publicados
  - - Total de reacciones recibidas
  - - Total de comentarios recibidos
  - Filtros y ordenamiento (más recientes, más populares, etc.)

# REQUERIMIENTOS TÉCNICOS

## Seguridad y Validaciones

- **Validaciones completas** tanto en frontend como en backend
- **CSRF deshabilitado** en Rails (usando solo JWT)
- **CORS configurado** para aceptar solo el dominio del frontend desplegado
- **Sanitización de inputs** para prevenir XSS e inyección SQL
- **Contraseñas encriptadas** con bcrypt
- Manejo seguro de tokens JWT en headers HTTP (no en localStorage para mayor seguridad)
- **Rate limiting** en endpoints críticos (registro, login, creación de posts)

## Testing

Se espera cobertura de testing en áreas críticas del sistema:

- **Backend (Rails + RSpec):** Tests de modelos, controladores, servicios y canales WebSocket
- **Frontend (Next.js + Jest/RTL):** Tests de componentes clave, hooks personalizados y flujos de usuario
- **Tests de integración:** Verificar flujos completos con autenticación JWT
- Mínimo recomendado: **70% de cobertura en backend**, tests de componentes críticos en frontend

## Dockerización (Solo Backend)

**Importante:** Solo el backend debe estar dockerizado. El frontend se despliega nativamente en Vercel/Netlify.

- **Docker Compose** configurado para el backend Rails
- Servicios: **web** (API Rails), **db** (PostgreSQL), **redis** (opcional)
- Volúmenes para persistencia de datos de la base de datos
- Variables de entorno gestionadas con archivo **.env.example**
- Dockerfile optimizado con multi-stage build
- Instrucciones claras en README para ejecutar **docker-compose up**
- **Frontend sin Docker:** Se ejecuta con **npm run dev** localmente y se despliega nativamente en Vercel/Netlify

## Optimización y Performance

- **Índices en PostgreSQL** para búsquedas, foreign keys y campos frecuentemente consultados
- **Eager loading** en Rails para evitar N+1 queries
- **Paginación eficiente** con cursor-based o offset-based según corresponda
- **Lazy loading de imágenes** en el frontend con Next.js Image
- **Caché de consultas frecuentes** con Redis (opcional)
- Optimización de bundle de JavaScript (code splitting automático en Next.js 15)
- Server Components de Next.js donde sea apropiado para mejor performance

## ENTREGABLES

- **Repositorio en GitHub** con código fuente completo y organizado
- **README.md detallado** con:
  - - Descripción del proyecto
  - - Instrucciones de instalación y ejecución (frontend y backend por separado)
  - - Variables de entorno necesarias para cada parte
  - - Comandos para correr tests
  - - Tecnologías utilizadas y decisiones arquitectónicas
  - - Instrucciones de deploy para cada servicio
- **Documentación técnica** en /docs/ incluyendo:
  - - Diagrama de arquitectura de despliegue
  - - Modelo de datos (ERD)
  - - Decisiones de diseño importantes
  - - Flujo de comunicación entre frontend y backend
- **Demo funcional desplegada (requerido):**
  - - Frontend en Vercel o Netlify
  - - Backend en Render.com o Fly.io
  - - URLs públicas funcionales para ambos servicios
- **Suite de tests** ejecutable con comandos documentados
- **Archivo Postman/Insomnia** con colección de endpoints de la API
- **Dockerfile y docker-compose.yml** funcional para el backend

## CRITERIOS DE EVALUACIÓN

Criterio	Peso	Descripción
<b>Funcionalidad</b>	30%	Todas las características requeridas funcionan correctamente
<b>Calidad de código</b>	25%	Código limpio, organizado, siguiendo mejores prácticas
<b>Arquitectura</b>	20%	Diseño escalable, modular y mantenable con deploy correcto
<b>Testing</b>	10%	Cobertura adecuada de tests unitarios e integración
<b>Documentación</b>	10%	README claro, comentarios útiles, documentación técnica

UX/UI	5%	Interfaz intuitiva, responsive y agradable visualmente
-------	----	--

## PUNTOS EXTRA (OPCIONAL)

Los siguientes elementos no son obligatorios, pero serán altamente valorados:

- ■ **Deploy funcional** con URLs públicas y funcionando correctamente (frontend + backend)
- ■ **Dashboard de admin** con estadísticas y moderación de contenido
- ■ **Animaciones sutiles** con Framer Motion o similar
- ■ **Envío real de emails** (no solo logs) para confirmación y recuperación
- ■ **Notificaciones push del navegador** usando Web Push API
- ■ **Accesibilidad** (ARIA labels, navegación por teclado, contraste adecuado)
- ■ **PWA** (Progressive Web App) con soporte offline
- ■ **CI/CD** configurado con GitHub Actions para deploy automático
- ■ **Monitoring y logs** con herramientas como Sentry o LogRocket
- ■ **E2E testing** con Cypress o Playwright
- ■ **HTTPS configurado** correctamente en ambos servicios
- ■ **Optimización avanzada** con Server Components y Streaming en Next.js 15

## RECOMENDACIONES Y CONSEJOS

- **Planificación inicial:** Dedica 4-6 horas a diseñar la arquitectura, el modelo de datos y la estructura de componentes antes de escribir código
- **Commits frecuentes:** Realiza commits descriptivos y atómicos. Esto demuestra tu proceso de trabajo y facilita la revisión
- **Prioriza funcionalidades core:** Implementa primero autenticación, CRUD de posts y sistema de comentarios. Las funcionalidades avanzadas pueden agregarse después
- **Deploy temprano:** Despliega el backend en Render/Fly.io y el frontend en Vercel/Netlify desde el inicio para validar la arquitectura
- **Configura CORS correctamente:** Asegúrate de que el backend acepte peticiones desde el dominio de tu frontend desplegado
- **Variables de entorno:** Documenta todas las variables necesarias en archivos .env.example
- **No reinventes la rueda:** Usa librerías establecidas para tareas comunes
- **Mobile-first:** Diseña pensando primero en dispositivos móviles y luego escala a desktop
- **Manejo de errores:** Implementa manejo robusto de errores tanto en frontend como backend
- **Feedback al usuario:** Muestra loaders, toasts y mensajes claros para cada acción
- **Consulta dudas:** Si algo no está claro en los requerimientos, es mejor preguntar que asumir

## CRONOGRAMA SUGERIDO (7 DÍAS)

Día	Tareas sugeridas
1-2	Setup inicial del proyecto (frontend y backend), configuración de Docker para backend, implementación de autenticación y CRUD de Posts.
3-4	CRUD de Posts, sistema de comentarios, reacciones y WebSockets. Configurar CORS correctamente.
5	Búsqueda avanzada, sistema de etiquetas, notificaciones en tiempo real.
6	Testing, optimizaciones con Next.js 15, modo oscuro, internacionalización.
7	Documentación completa, verificación de deploy funcional, refinamiento de UI/UX, preparación de entrega.

## NOTAS FINALES

Esta prueba técnica está diseñada para evaluar tus habilidades técnicas integrales como desarrollador fullstack. No se espera perfección en todos los aspectos, pero sí una

demostración clara de tus capacidades en diseño de arquitectura, implementación de funcionalidades, manejo de estado, optimización y mejores prácticas de desarrollo.

**Puntos clave a recordar:**

- Usa **Next.js 15** (última versión) para el frontend
- Solo el **backend debe estar dockerizado**
- El deploy es **requerido** y debe funcionar correctamente
- Frontend en **Vercel o Netlify**, Backend en **Render.com o Fly.io**
- Configura **CORS** correctamente para la comunicación entre servicios

**¡Éxito en tu prueba técnica!** Si tienes preguntas durante el desarrollo, no dudes en contactarnos.