FASE 3: Arquitectura de Sincronización y Tiempo Real

Hablando de Caballos - Sistema Unificado Web/Móvil

STACK TECNOLÓGICO

- **WebSockets**: Socket.io con Redis Adapter para escalabilidad
- **Cache/Sesiones**: Redis para almacenamiento distribuido
- **Push Notifications**: Service Workers + Web Push API
- **Real-time Sync**: Eventos bidireccionales entre clientes
- **Performance**: Lazy loading, infinite scroll, compresión de imágenes

ARQUITECTURA DEL SISTEMA

1. SERVIDOR WEBSOCKET

```
Custom Next.js Server (server.js)
├── Socket.io Server con Redis Adapter
├── Eventos en tiempo real
├── Gestión de salas/rooms
└── Broadcasting distribuido
```

2. REDIS LAYER

```
Redis Instance
├── Session Store (express-session + connect-redis)
├── Cache de consultas DB
├── Pub/Sub para WebSockets
└── Datos temporales de usuario
```

3. SISTEMA DE NOTIFICACIONES

```
Push Notifications
├── Service Worker (public/sw.js)
├── Web Push API + VAPID keys
├── Subscription management
└── Notification types: comentarios, menciones, votos
```

4. SINCRONIZACIÓN DE DATOS

```
Real-time Events
├── Nuevos posts/comentarios
├── Votos y reacciones
├── Estado de usuarios online
└── Notificaciones instantáneas
```

EVENTOS WEBSOCKET DEFINIDOS

- `user:online` - Usuario conectado

- `user:offline` - Usuario desconectado
- `post:new` - Nuevo post creado
- `comment:new` - Nuevo comentario
- `vote:update` - Actualización de votos
- `notification:push` - Notificación push
- `room:join` - Unirse a sala (foro específico)
- `room:leave` - Salir de sala

FLUJO DE IMPLEMENTACIÓN

1. Configurar servidor WebSocket personalizado
2. Integrar Redis para cache y sesiones
3. Implementar Service Worker y push notifications
4. Crear hooks React para eventos en tiempo real
5. Optimizar performance con lazy loading
6. Añadir monitoreo y analytics

COMPATIBILIDAD

- Next.js 14+ con App Router
- Redis 7.0+ para sharded pub/sub
- Navegadores modernos con Service Worker support
- HTTPS requerido para push notifications en producción