Documentación de Prioridades y Segunda Iteración - Qamarero

📋 Resumen del Proyecto

Qamarero es un sistema de gestión de pagos para restaurantes que permite a los camareros dividir y procesar pagos de pedidos de múltiples formas. El sistema soporta tres modos principales de pago:

1. **Pagar Todo** - Pago único de toda la cuenta
2. **Dividir en Partes Iguales** - División automática del total entre grupos
3. **Personalizar** - Asignación manual de items específicos a grupos

---

🎯 ¿Qué se Priorizó en la Primera Iteración?

1. **Arquitectura y Stack Tecnológico**

- **Monorepo con Turborepo**: Estructura modular que separa claramente la lógica de negocio (`packages/api`), la base de datos (`packages/db`) y la interfaz (`apps/web`)
- **TypeScript end-to-end**: Tipado completo desde la base de datos hasta el frontend usando tRPC y Drizzle ORM
- **Stack moderno**: Next.js 15 (App Router), React Query, TailwindCSS, shadcn/ui
- **Razón**: Priorizar la seguridad de tipos y escalabilidad desde el inicio

2. **Funcionalidades Core del Negocio**

- **Sistema de pagos flexible**: Modal de pago reutilizable que soporta múltiples métodos (efectivo, tarjeta)
- **Tres modos de división**: Cubrir los casos de uso más comunes
- **Gestión de estado de pagos**: Tracking acumulativo de pagos parciales por método y por grupo
- **Razón**: Resolver el problema principal (división de cuentas) de forma completa y funcional

3. **Experiencia de Usuario (UX)**

- **Interfaz responsive**: Diseño adaptativo para diferentes tamaños de pantalla
- **Feedback visual claro**: Indicadores de estado de pago, montos pendientes, y estados de carga/error
- **Modal de pago intuitivo**: Validaciones en tiempo real, cálculos automáticos de montos pendientes

4. **Organización del Código**

- **Componentes modulares**: Separación de responsabilidades (header, content, items, groups, etc.)
- **Razón**: Facilitar el mantenimiento y la extensión futura

5. **Funcionalidad de Personalización Avanzada**

- **Asignación granular de items**: Permitir asignar cantidades específicas de items a grupos
- **Diálogo de asignación**: Selección múltiple de items y grupos para asignación masiva
- **Validaciones de cantidades**: Prevenir sobre-asignación de items

- **Razón**: Cubrir casos de uso complejos donde la división igual no aplica

---

🔄 ¿Qué se Haría Distinto en una Segunda Iteración?

1. **Persistencia de Datos y Estado**

Problema actual: El estado de pagos y asignaciones de grupos solo existe en memoria del cliente. Si se recarga la página, se pierde todo.

Mejora:

- Agregar tablas de base de datos para:
  - Registrar pagos realizados (método, monto, grupo, timestamp)
  - Persistir los grupos creados y sus asignaciones
  - Relacionar items con grupos en divisiones personalizadas
- Implementar endpoints tRPC para:
  - Guardar configuración de división
  - Registrar pagos realizados
  - Recuperar historial de pagos
- Agregar estado de "guardado"/"borrador" para permitir continuar trabajando

Impacto: Permitiría recuperar el trabajo en caso de errores, revisar historial, y generar reportes.

---

2. **Rendimiento y Optimización**

Problema actual: Posibles re-renders innecesarios y cálculos repetidos.

Mejora:

- Memorización de cálculos pesados:
  - `useMemo` para totales y cálculos de grupos
  - `useCallback` para handlers que se pasan a múltiples componentes
- Optimización de queries:
  - Cache más agresivo de datos de pedidos
  - Invalidación inteligente de cache solo cuando sea necesario
- Code splitting:
  - Lazy loading de modales y diálogos
  - Carga diferida de componentes pesados
- Optimización de renders:
  - Revisar si se pueden dividir componentes grandes en más pequeños
  - Usar React.memo donde tenga sentido

Impacto: Mejor rendimiento en dispositivos menos potentes y mejor experiencia general.

---

3. **Funcionalidades Faltantes**

Problema actual: Funcionalidades básicas para un sistema completo de pagos.

Mejora:

- **Historial y auditoría**:
  - Ver historial de pagos de una mesa
- **Impresión y tickets**:
  - Generar tickets de pago por grupo
  - Imprimir resumen completo de la mesa
- **Gestión de descuentos y propinas**:
  - Aplicar descuentos a nivel de item o total
  - Distribución automática de propinas
- **Notificaciones en tiempo real**:
  - Actualizaciones cuando otro camarero procesa un pago
  - Notificaciones cuando se completa el pago total

Impacto: Sistema más completo y profesional para uso en producción.

---

4. **Mejoras en la Interfaz de Usuario**

Problema actual: UI funcional pero con margen de mejora en UX y accesibilidad.

Mejora:

- **Accesibilidad**:
  - Navegación por teclado en todos los modales
  - Soporte para lectores de pantalla
- **Atajos de teclado**:
  - Atajos rápidos para métodos de pago (E = Efectivo, T = Tarjeta)
  - Navegación rápida entre grupos
- **Feedback mejorado**:
  - Animaciones de transición más suaves
  - Confirmaciones antes de acciones destructivas
  - Loading states más informativos
- **Temas y personalización**:
  - Diferentes temas visuales
  - Personalización de colores por establecimiento
- **Responsive mejorado**:
  - Mejor experiencia en tablets
  - Modo landscape optimizado

Impacto: Mejor usabilidad y accesibilidad para todos los usuarios.

---

5. **Arquitectura y Escalabilidad**

Problema actual: Arquitectura buena pero con mejoras posibles para escalar.

Mejora:

- **Separación de lógica de negocio**:

    - Mover más lógica de cálculo al backend (tRPC)
    - Validaciones de negocio en el servidor, no solo en el cliente
- **Estado global**:
    - Considerar Zustand o Jotai para estado compartido complejos
- **API más robusta**:
    - Versionado de API
    - Rate limiting
    - Autenticación y autorización (si aplica multi-usuario)
- **Base de datos**:
    - Índices optimizados para queries frecuentes
    - Soft deletes para mantener historial
    - Migraciones versionadas

📊 Priorización para Segunda Iteración (Orden Sugerido)

1. **Persistencia de datos**
2. **Validaciones y manejo de errores en backend**
3. **Testing básico** (Alto) - Prevenir regresiones
4. **Mejoras de UX/UI** (Medio) - Mejora experiencia diaria
5. **Funcionalidades de historial/auditoría** (Medio)
6. **Rendimiento** (Bajo-Medio) - Optimizar cuando haya datos reales

---

🎬 Conclusión

La primera iteración priorizó correctamente:

- ✅ Establecer una base sólida y tipada
- ✅ Implementar las funcionalidades core del negocio
- ✅ Crear una experiencia de usuario funcional

Para la segunda iteración, el foco debe estar en:

- 🔒 **Robustez**: Persistencia, validaciones, tests
- 📈 **Escalabilidad**: Arquitectura mejorada, rendimiento
- 🎨 **Pulido**: UX mejorada, documentación, funcionalidades adicionales