

Guía de Integración Backend - Módulo Dashboard

Versión: 1.0.0 **Última Actualización:** 2025-10-01 **Estado:** ● Datos Stub
(Hardcodeados) → ● Integración Backend Requerida

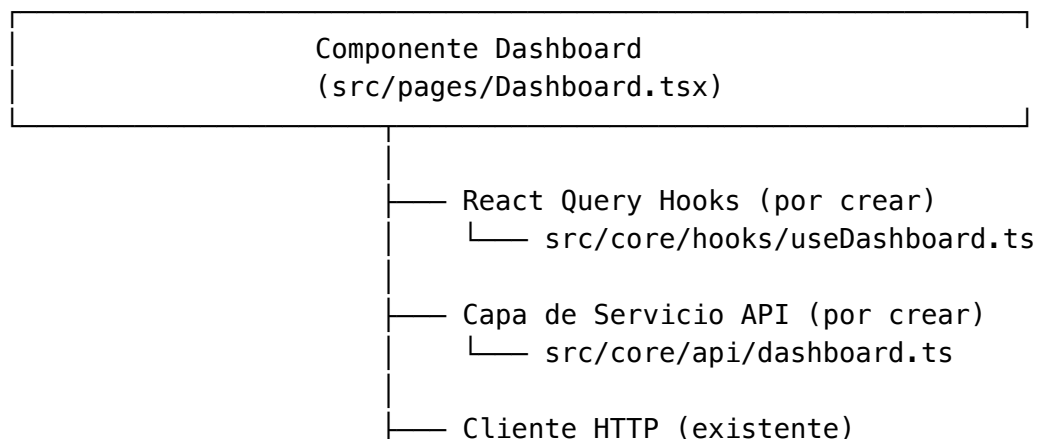
Resumen General

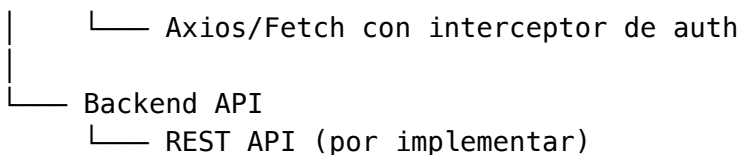
Este documento describe los requisitos completos de integración backend para el Dashboard del Portal Pazz. Todos los datos actuales están **hardcodeados usando constantes STUB_** en `src/pages/Dashboard.tsx`. Esta guía proporciona la hoja de ruta para conectar con una API backend real.

Estado de Implementación Actual

- ✔ Componentes UI: Completamente implementados
 - ✔ Definiciones de Tipos: Completas (`src/types/dashboard.ts`)
 - ✔ Interacciones de Modales: Funcionando con datos stub
 - ● Integración API: **No implementada** (usando datos hardcodeados)
 - ● Actualizaciones en Tiempo Real: No implementadas
 - ● Manejo de Errores: Solo básico del lado del cliente
-

Visión General de Arquitectura





Especificación de Endpoints API

1. Perfil de Usuario y Sistema de Niveles

GET /api/v1/user/profile

Propósito: Obtener el perfil del usuario actual incluyendo nivel y datos de ventas

Autenticación: Requerida (Bearer token)

Solicitud:

GET /api/v1/user/profile
Authorization: Bearer {token}

Respuesta:

```
{
  "success": true,
  "data": {
    "userId": "usr_abc123",
    "currentLevel": "starter" | "master" | "legend",
    "accumulatedSales": 4020000,
    "levelProgress": {
      "currentSales": 4020000,
      "nextLevelRequired": 4500000,
      "percentageComplete": 89.3
    },
    "commissionRate": 2.0
  }
}
```

Respuestas de Error: - 401 Unauthorized: Token inválido o expirado - 403 Forbidden: El usuario no tiene acceso - 500 Internal Server Error: Error del servidor

Estrategia de Caché: - Caché por 5 minutos (staleTime: 5 * 60 * 1000) - Invalidar en actualizaciones de comisiones

2. Gestión de Metas

GET /api/v1/goals/current

Propósito: Obtener la meta del mes actual y su progreso

Solicitud:

GET /api/v1/goals/current
Authorization: Bearer {token}

Respuesta:

```
{
  "success": true,
  "data": {
    "id": "goal_xyz789",
    "amount": 25000,
    "currentAmount": 20000,
    "month": "Octubre",
    "year": 2025,
    "achieved": false,
    "progress": 80.0,
    "remaining": 5000,
    "earnedCommissions": 20000
  }
}
```

Lógica de Negocio: - Las metas son mensuales (se reinician el día 1 de cada mes) -
currentAmount se actualiza automáticamente basado en comisiones ganadas - Bandera
achieved se establece en true cuando currentAmount >= amount

GET /api/v1/goals/history

Propósito: Obtener metas pasadas con estado de logro

Solicitud:

GET /api/v1/goals/history?limit=12&offset=0
Authorization: Bearer {token}

Parámetros de Consulta: - limit (opcional): Número de registros (default: 12, max: 24) - offset (opcional): Offset de paginación (default: 0)

Respuesta:

```
{
  "success": true,
  "data": {
    "goals": [
      {
        "id": "goal_456",
```

```

    "amount": 25000,
    "currentAmount": 26500,
    "month": "Mayo",
    "year": 2025,
    "achieved": true,
    "progress": 106.0,
    "exceededBy": 1500
  }
],
"pagination": {
  "total": 8,
  "limit": 12,
  "offset": 0,
  "hasMore": false
}
}
}

```

Estrategia de Caché: - Caché por 1 hora (datos históricos raramente cambian) - Invalidar cuando se crea una nueva meta

POST /api/v1/goals

Propósito: Crear o actualizar meta mensual

Solicitud:

POST /api/v1/goals
 Authorization: Bearer {token}
 Content-Type: application/json

```

{
  "amount": 30000,
  "month": "Noviembre",
  "year": 2025
}

```

Respuesta:

```

{
  "success": true,
  "data": {
    "id": "goal_new123",
    "amount": 30000,
    "currentAmount": 0,
    "month": "Noviembre",
    "year": 2025,
    "achieved": false,
    "createdAt": "2025-10-01T10:30:00Z"
  }
}

```

Reglas de Validación: - amount: Requerido, número positivo, min: 1000, max: 1000000
- month: Requerido, nombre de mes válido (español) - year: Requerido, año actual o futuro - Solo una meta por mes permitida (comportamiento upsert)

Respuestas de Error: - 400 Bad Request: Entrada inválida (con errores a nivel de campo) - 409 Conflict: Ya existe una meta para este mes

Estrategia de Actualización Optimista: - Actualizar UI inmediatamente antes de llamada API - Rollback en error con notificación toast

3. Sistema de Promociones

GET /api/v1/promotions/active

Propósito: Obtener promociones actualmente activas

Solicitud:

GET /api/v1/promotions/active
Authorization: Bearer {token}

Respuesta:

```
{
  "success": true,
  "data": {
    "promotions": [
      {
        "id": "promo_bmw001",
        "title": "BMW Serie 4",
        "subtitle": "Gran Coupé 2025",
        "imageUrl": "https://cdn.pazz.com/promotions/bmw-serie4.jpg",
        "bonusAmount": 100000,
        "originalPrice": 1245000,
        "finalPrice": 1145000,
        "validFrom": "2025-08-01T00:00:00Z",
        "validTo": "2025-08-31T23:59:59Z",
        "isActive": true,
        "monthlyPayments": [
          { "months": 48, "amount": 45000 },
          { "months": 36, "amount": 47120 },
          { "months": 24, "amount": 49785 }
        ],
        "termsAndConditions": "https://pazz.com/terms/promo_bmw001"
      }
    ],
    "metadata": {
      "totalActive": 1,
      "nextPromotion": null
    }
  }
}
```

```
}
}
```

Lógica de Negocio: - Las promociones son a nivel sistema (visibles para todos los socios) - Solo se retornan promociones activas (fecha actual dentro de validFrom/validTo) - Ordenadas por prioridad/fecha validFrom - URLs de imagen deben estar optimizadas por CDN

Estrategia de Caché: - Caché por 1 hora (las promociones no cambian frecuentemente) - Refresco en segundo plano cada 15 minutos - Invalidación manual desde panel de admin

4. Sistema de Seguimiento de Clientes

GET /api/v1/clients/attention-required

Propósito: Obtener clientes que necesitan acción de seguimiento

Solicitud:

GET /api/v1/clients/attention-required?limit=20&status=all
Authorization: Bearer {token}

Parámetros de Consulta: - limit (opcional): Máximo de resultados (default: 20, max: 50) - status (opcional): Filtrar por estado: prequalification, documentation, approval, all (default: all)

Respuesta:

```
{
  "success": true,
  "data": {
    "clients": [
      {
        "id": "client_abc123",
        "name": "Pedro Juan Gutiérrez González",
        "status": "prequalification",
        "statusLabel": "Precalificación sin concluir",
        "lastAccessed": "2025-08-14T15:30:00Z",
        "startedAt": "2025-08-12T10:00:00Z",
        "phone": "5215598772310",
        "email": "pedro.gutierrez@example.com",
        "message": "Hola Pedro, vi que iniciaste tu precalificación pero no la has completado...",
        "daysInactive": 3,
        "progressPercentage": 45,
        "nextAction": "complete_prequalification",
        "priority": "high" | "medium" | "low"
      }
    ],
    "metadata": {
      "total": 3,

```

```

    "byStatus": {
      "prequalification": 2,
      "documentation": 1,
      "approval": 0,
      "financing": 0
    }
  }
}
}
}

```

Lógica de Negocio: - Solo retorna clientes asignados al socio autenticado - Ordenados por prioridad, luego por lastAccessed (más antiguo primero) - Lógica de “Requiere atención”: - Precalificación incompleta > 48 horas - Documentación incompleta > 72 horas - Sin actividad en los últimos 7 días - Los mensajes se generan previamente basados en estado e inactividad

Estrategia de Caché: - Caché por 2 minutos (datos cambian frecuentemente) - Polling cada 5 minutos para actualizaciones - Actualizaciones en tiempo real vía WebSocket (opcional)

Cálculo de Prioridad: - high: Inactivo > 7 días O negocio de alto valor (>500k) - medium: Inactivo 3-7 días - low: Inactivo < 3 días

GET /api/v1/clients/:clientId

Propósito: Obtener información detallada del cliente

Solicitud:

GET /api/v1/clients/client_abc123
 Authorization: Bearer {token}

Respuesta:

```

{
  "success": true,
  "data": {
    "id": "client_abc123",
    "name": "Pedro Juan Gutiérrez González",
    "email": "pedro.gutierrez@example.com",
    "phone": "5215598772310",
    "status": "prequalification",
    "statusLabel": "Precalificación sin concluir",
    "startedAt": "2025-08-12T10:00:00Z",
    "lastAccessed": "2025-08-14T15:30:00Z",
    "progressPercentage": 45,
    "timeline": [
      {
        "event": "prequalification_started",
        "timestamp": "2025-08-12T10:00:00Z",
        "description": "Inició precalificación"
      },
    ]
  }
}

```

```

    {
      "event": "document_uploaded",
      "timestamp": "2025-08-12T10:15:00Z",
      "description": "Subió INE"
    }
  ],
  "vehicleInterest": {
    "make": "BMW",
    "model": "Serie 4",
    "year": 2025,
    "estimatedValue": 1245000
  },
  "assignedPartner": {
    "id": "usr_partner123",
    "name": "Current User"
  }
}
}

```

5. Comisiones y Ganancias

GET /api/v1/commissions/summary

Propósito: Obtener resumen de comisiones para el mes actual

Solicitud:

GET /api/v1/commissions/summary?month=10&year=2025

Authorization: Bearer {token}

Parámetros de Consulta: - month (opcional): Número de mes (1-12, default: mes actual) - year (opcional): Año (default: año actual)

Respuesta:

```

{
  "success": true,
  "data": {
    "period": {
      "month": "Octubre",
      "year": 2025,
      "monthNumber": 10
    },
    "earned": {
      "amount": 20000,
      "currency": "MXN",
      "deals": 5
    },
    "pending": {
      "amount": 15000,
      "currency": "MXN",

```



```

    "deals": 3
  },
  "breakdown": [
    {
      "dealId": "deal_001",
      "clientName": "Juan Pérez",
      "vehicleMake": "BMW",
      "vehicleModel": "Serie 4",
      "commissionAmount": 5000,
      "commissionRate": 2.0,
      "status": "paid",
      "paidAt": "2025-10-05T10:00:00Z"
    }
  ]
}
}
}

```

Estrategia de Caché: - Caché por 5 minutos - Invalidar en completado de nuevo negocio

Modelos de Datos (Interfaces TypeScript)

Wrapper de Respuesta API

```
// src/types/api.ts
```

```

export interface ApiResponse<T> {
  success: boolean
  data: T
  error?: {
    code: string
    message: string
    details?: Record<string, string[]>
  }
  metadata?: {
    timestamp: string
    requestId: string
  }
}

export interface PaginatedResponse<T> {
  items: T[]
  pagination: {
    total: number
    limit: number
    offset: number
    hasMore: boolean
  }
}

```

Tipos Específicos del Dashboard

```
// src/types/dashboard-api.ts
```

```
export interface UserProfileResponse {  
  userId: string  
  currentLevel: LevelType  
  accumulatedSales: number  
  levelProgress: {  
    currentSales: number  
    nextLevelRequired: number  
    percentageComplete: number  
  }  
  commissionRate: number  
}
```

```
export interface GoalResponse {  
  id: string  
  amount: number  
  currentAmount: number  
  month: string  
  year: number  
  achieved: boolean  
  progress: number  
  remaining: number  
  earnedCommissions?: number  
}
```

```
export interface CreateGoalRequest {  
  amount: number  
  month: string  
  year: number  
}
```

```
export interface PromotionResponse {  
  id: string  
  title: string  
  subtitle: string  
  imageUrl: string  
  bonusAmount: number  
  originalPrice: number  
  finalPrice: number  
  validFrom: string  
  validTo: string  
  isActive: boolean  
  monthlyPayments: Array<{  
    months: number  
    amount: number  
  }>  
  termsAndConditions?: string  
}
```

```
export interface ClientAttentionResponse {
  id: string
  name: string
  status: ClientStatus
  statusLabel: string
  lastAccessed: string
  startedAt: string
  phone?: string
  email?: string
  message?: string
  daysInactive: number
  progressPercentage: number
  nextAction: string
  priority: 'high' | 'medium' | 'low'
}
```

Guía de Implementación

Paso 1: Crear Capa de Servicio API

```
// src/core/api/dashboard.ts
```

```
import type {
  UserProfileResponse,
  GoalResponse,
  CreateGoalRequest,
  PromotionResponse,
  ClientAttentionResponse,
} from '@types/dashboard-api'
import type { ApiResponse, PaginatedResponse } from '@types/api'
import { apiClient } from './client' // Instancia de Axios con auth

export const dashboardApi = {
  // Perfil de Usuario y Nivel
  getUserProfile: async (): Promise<UserProfileResponse> => {
    const response = await
      apiClient.get<ApiResponse<UserProfileResponse>>(
        '/api/v1/user/profile'
      )
    return response.data.data
  },

  // Metas
  getCurrentGoal: async (): Promise<GoalResponse> => {
    const response = await apiClient.get<ApiResponse<GoalResponse>>(
      '/api/v1/goals/current'
    )
  }
}
```

```

    return response.data.data
  },

  getGoalHistory: async (limit = 12): Promise<GoalResponse[]> => {
    const response = await apiClient.get<
      ApiResponse<PaginatedResponse<GoalResponse>>
    >('/api/v1/goals/history', { params: { limit } })
    return response.data.data.items
  },

  createGoal: async (goal: CreateGoalRequest): Promise<GoalResponse>
    => {
    const response = await apiClient.post<ApiResponse<GoalResponse>>(
      '/api/v1/goals',
      goal
    )
    return response.data.data
  },

  // Promociones
  getActivePromotions: async (): Promise<PromotionResponse[]> => {
    const response = await apiClient.get<
      ApiResponse<{ promotions: PromotionResponse[] }>
    >('/api/v1/promotions/active')
    return response.data.data.promotions
  },

  // Clientes
  getClientsNeedingAttention: async ():
    Promise<ClientAttentionResponse[]> => {
    const response = await apiClient.get<
      ApiResponse<{ clients: ClientAttentionResponse[] }>
    >('/api/v1/clients/attention-required')
    return response.data.data.clients
  },
}

```

Paso 2: Crear Hooks de React Query

```

// src/core/hooks/useDashboard.ts

import { useQuery, useMutation, useQueryClient } from
  '@tanstack/react-query'
import { dashboardApi } from '@core/api/dashboard'
import type { CreateGoalRequest } from '@types/dashboard-api'
import { toast } from 'sonner'

// Query Keys
export const DASHBOARD_KEYS = {
  profile: ['dashboard', 'profile'] as const,

```

```
currentGoal: ['dashboard', 'goals', 'current'] as const,  
goalHistory: ['dashboard', 'goals', 'history'] as const,  
promotions: ['dashboard', 'promotions'] as const,  
clientsAttention: ['dashboard', 'clients', 'attention'] as const,  
}
```

// Hook de Perfil de Usuario

```
export const useUserProfile = () => {  
  return useQuery({  
    queryKey: DASHBOARD_KEYS.profile,  
    queryFn: dashboardApi.getUserProfile,  
    staleTime: 5 * 60 * 1000, // 5 minutos  
    gcTime: 10 * 60 * 1000, // 10 minutos  
    retry: 2,  
  })  
}
```

// Hook de Meta Actual

```
export const useCurrentGoal = () => {  
  return useQuery({  
    queryKey: DASHBOARD_KEYS.currentGoal,  
    queryFn: dashboardApi.getCurrentGoal,  
    staleTime: 2 * 60 * 1000, // 2 minutos  
    retry: 2,  
  })  
}
```

// Hook de Historial de Metas

```
export const useGoalHistory = () => {  
  return useQuery({  
    queryKey: DASHBOARD_KEYS.goalHistory,  
    queryFn: () => dashboardApi.getGoalHistory(),  
    staleTime: 60 * 60 * 1000, // 1 hora (datos históricos)  
    retry: 1,  
  })  
}
```

// Mutación de Crear Meta

```
export const useCreateGoal = () => {  
  const queryClient = useQueryClient()  
  
  return useMutation({  
    mutationFn: (goal: CreateGoalRequest) =>  
      dashboardApi.createGoal(goal),  
    onSuccess: (data) => {  
      // Invalidar consultas relacionadas  
      queryClient.invalidateQueries({ queryKey:  
        DASHBOARD_KEYS.currentGoal })  
      queryClient.invalidateQueries({ queryKey:  
        DASHBOARD_KEYS.goalHistory })  
      toast.success('Meta guardada exitosamente')  
    },  
  })  
}
```

```

    onError: (error: any) => {
      toast.error(error.response?.data?.error?.message || 'Error al
        guardar la meta')
    },
  })
}

// Hook de Promociones Activas
export const useActivePromotions = () => {
  return useQuery({
    queryKey: DASHBOARD_KEYS.promotions,
    queryFn: dashboardApi.getActivePromotions,
    staleTime: 60 * 60 * 1000, // 1 hora
    gcTime: 2 * 60 * 60 * 1000, // 2 horas
    retry: 2,
  })
}

// Hook de Clientes que Requieren Atención
export const useClientsNeedingAttention = () => {
  return useQuery({
    queryKey: DASHBOARD_KEYS.clientsAttention,
    queryFn: dashboardApi.getClientsNeedingAttention,
    staleTime: 2 * 60 * 1000, // 2 minutos
    refetchInterval: 5 * 60 * 1000, // Polling cada 5 minutos
    retry: 2,
  })
}

```

Paso 3: Actualizar Componente Dashboard

// src/pages/Dashboard.tsx – VERSIÓN ACTUALIZADA

```

import React, { useState } from 'react'
import { useOutletContext } from 'react-router-dom'
import { LevelBadge } from '@components/dashboard/LevelBadge'
import { GoalProgressCard } from
  '@components/dashboard/GoalProgressCard'
import { PromotionCard } from '@components/dashboard/PromotionCard'
import { ClientAttentionItem } from
  '@components/dashboard/ClientAttentionItem'
import { GoalModal } from '@components/dashboard/GoalModal'
import { PromotionModal } from '@components/dashboard/PromotionModal'
import { ClientDetailModal } from
  '@components/dashboard/ClientDetailModal'
import { LevelModal } from '@components/dashboard/LevelModal'
import { Spinner } from '@ui/spinner'
import type { Promotion, Client } from '@types/dashboard'

// Importar hooks de React Query
import {

```

```

    useUserProfile,
    useCurrentGoal,
    useGoalHistory,
    useActivePromotions,
    useClientsNeedingAttention,
    useCreateGoal,
  } from '@core/hooks/useDashboard'

interface DashboardOutletContext {
  setHeaderRightElement?: (element: React.ReactNode) => void
}

export default function Dashboard() {
  const { setHeaderRightElement } =
    useOutletContext<DashboardOutletContext>()

  // Hooks de React Query
  const { data: profile, isLoading: profileLoading } =
    useUserProfile()
  const { data: currentGoal, isLoading: goalLoading } =
    useCurrentGoal()
  const { data: goalHistory } = useGoalHistory()
  const { data: promotions, isLoading: promotionsLoading } =
    useActivePromotions()
  const { data: clients, isLoading: clientsLoading } =
    useClientsNeedingAttention()
  const createGoalMutation = useCreateGoal()

  // Estados de modales
  const [goalModalOpen, setGoalModalOpen] = useState(false)
  const [promotionModalOpen, setPromotionModalOpen] = useState(false)
  const [clientModalOpen, setClientModalOpen] = useState(false)
  const [levelModalOpen, setLevelModalOpen] = useState(false)
  const [selectedPromotion, setSelectedPromotion] = useState<Promotion
    | null>(null)
  const [selectedClient, setSelectedClient] = useState<Client | null>
    (null)

  // Establecer badge de nivel en header
  React.useEffect(() => {
    if (profile) {
      setHeaderRightElement?.(
        <LevelBadge
          level={profile.currentLevel}
          onClick={() => setLevelModalOpen(true)}
        />
      )
    }
  }, [profile, setHeaderRightElement])

  return () => setHeaderRightElement?.(null)
}, [profile, setHeaderRightElement])

// Manejar guardar meta
const handleSaveGoal = (amount: number) => {

```

```

    createGoalMutation.mutate({
      amount,
      month: new Date().toLocaleString('es-MX', { month: 'long' }),
      year: new Date().getFullYear(),
    })
  }

  // Estado de carga
  if (profileLoading || goalLoading) {
    return (
      <div className="flex items-center justify-center h-64">
        <Spinner size="lg" />
      </div>
    )
  }

  // Estado de error (perfil y meta son requeridos)
  if (!profile || !currentGoal) {
    return (
      <div className="text-center text-slate-600 py-12">
        No se pudo cargar la información. Por favor, intenta de nuevo.
      </div>
    )
  }

  return (
    <div className="space-y-6 pb-6 pt-4">
      <GoalProgressCard
        goal={currentGoal}
        earnedCommissions={currentGoal.earnedCommissions || 0}
        onGoalClick={() => setGoalModalOpen(true)}
      />

      <div className="space-y-3">
        <h2 className="text-lg font-semibold">Promociones
          vigentes</h2>
        {promotionsLoading ? (
          <div className="flex gap-4">
            <div className="w-[280px] h-[300px] bg-slate-100 rounded-
              lg animate-pulse" />
          </div>
        ) : promotions && promotions.length > 0 ? (
          <div className="flex gap-4 overflow-x-auto pb-2 -mx-4 px-4
            sm:mx-0 sm:px-0">
            {promotions.map((promotion) => (
              <PromotionCard
                key={promotion.id}
                promotion={promotion}
                onClick={() => {
                  setSelectedPromotion(promotion)
                  setPromotionModalOpen(true)
                }}
              >
            )}
          </div>
        ) : null}
      </div>
    </div>
  )

```



```

    />
  )})
</div>
) : (
  <p className="text-sm text-slate-600">No hay promociones
  activas</p>
  )}
</div>

<div className="space-y-3">
  <h2 className="text-lg font-semibold">Clientes que requieren
  atención</h2>
  {clientsLoading ? (
    <div className="space-y-2">
      <div className="h-20 bg-slate-100 rounded-lg animate-
      pulse" />
      <div className="h-20 bg-slate-100 rounded-lg animate-
      pulse" />
    </div>
  ) : clients && clients.length > 0 ? (
    <div className="space-y-2">
      {clients.map((client) => (
        <ClientAttentionItem
          key={client.id}
          client={client}
          onClick={() => {
            setSelectedClient(client)
            setClientModalOpen(true)
          }}
        />
      )})
    </div>
  ) : (
    <p className="text-sm text-slate-600">
      No hay clientes que requieran atención
    </p>
  )}
</div>

{/* Modales */}
<GoalModal
  open={goalModalOpen}
  onOpenChange={setGoalModalOpen}
  currentGoal={currentGoal}
  pastGoals={goalHistory || []}
  onSaveGoal={handleSaveGoal}
/>

<PromotionModal
  open={promotionModalOpen}
  onOpenChange={setPromotionModalOpen}
  promotion={selectedPromotion}

```

```

    />

    <ClientDetailModal
      open={clientModalOpen}
      onOpenChange={setClientModalOpen}
      client={selectedClient}
    />

    <LevelModal
      open={levelModalOpen}
      onOpenChange={setLevelModalOpen}
      currentLevel={profile.currentLevel}
      accumulatedSales={profile.accumulatedSales}
    />
  </div>
)
}

```

Autenticación y Autorización

Estrategia de Bearer Token

Todas las solicitudes API deben incluir token de autenticación:

```

// src/core/api/client.ts

import axios from 'axios'
import { getAuthToken, refreshAuthToken } from '@core/auth'

export const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_BASE_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
})

// Interceptor de solicitud: Agregar token de auth
apiClient.interceptors.request.use(
  (config) => {
    const token = getAuthToken()
    if (token) {
      config.headers.Authorization = `Bearer ${token}`
    }
    return config
  },
  (error) => Promise.reject(error)
)

```

```

// Interceptor de respuesta: Manejar refresco de token
apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config

    // Token expirado - intentar refresco
    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true

      try {
        const newToken = await refreshAuthToken()
        originalRequest.headers.Authorization = `Bearer ${newToken}`
        return apiClient(originalRequest)
      } catch (refreshError) {
        // Refresco falló - redirigir a login
        window.location.href = '/auth/login'
        return Promise.reject(refreshError)
      }
    }

    return Promise.reject(error)
  }
)

```

Control de Acceso Basado en Roles

```

// El backend debe validar el acceso del socio a los datos del cliente
// Cada endpoint debe verificar:
// - El usuario está autenticado
// - El usuario tiene rol 'partner'
// - El usuario solo puede acceder a sus clientes asignados
// - El usuario solo puede ver promociones para su región/nivel

```



Patrones de Manejo de Errores

Respuesta de Error Estandarizada

```

// Todos los errores del backend deben seguir este formato
{
  "success": false,
  "error": {
    "code": "GOAL_VALIDATION_FAILED",
    "message": "La meta debe ser mayor a $1,000",
    "details": {
      "amount": ["Debe ser mayor a 1000"]
    }
  }
}

```

```
  },  
  "metadata": {  
    "timestamp": "2025-10-01T10:30:00Z",  
    "requestId": "req_abc123"  
  }  
}
```

Manejo de Errores en Frontend

```
// src/core/utils/error-handler.ts  
  
import { toast } from 'sonner'  
  
export const handleApiError = (error: any, defaultMessage: string) =>  
  {  
    if (error.response?.data?.error) {  
      const apiError = error.response.data.error  
  
      // Mostrar errores a nivel de campo  
      if (apiError.details) {  
        Object.entries(apiError.details).forEach(([field, messages]) =>  
          {  
            toast.error(`${field}: ${messages as string[]}.join(', ')`)  
          })  
      } else {  
        toast.error(apiError.message || defaultMessage)  
      }  
    } else if (error.message === 'Network Error') {  
      toast.error('Error de conexión. Verifica tu internet.')  
    } else {  
      toast.error(defaultMessage)  
    }  
  }  
}
```

Optimización de Rendimiento

1. Obtención de Datos en Paralelo

```
// Usar la función de consultas paralelas de React Query  
export const useDashboardData = () => {  
  const profile = useUserProfile()  
  const goal = useCurrentGoal()  
  const promotions = useActivePromotions()  
  const clients = useClientsNeedingAttention()  
  
  return {  
    isLoading: profile.isLoading || goal.isLoading,  
    isError: profile.isError || goal.isError,  
  }  
}
```

```

    data: {
      profile: profile.data,
      goal: goal.data,
      promotions: promotions.data,
      clients: clients.data,
    },
  },
}
}

```

2. Actualizaciones Optimistas

```

export const useCreateGoal = () => {
  const queryClient = useQueryClient()

  return useMutation({
    mutationFn: dashboardApi.createGoal,
    // Actualización optimista
    onMutate: async (newGoal) => {
      await queryClient.cancelQueries({ queryKey:
        DASHBOARD_KEYS.currentGoal })

      const previousGoal =
        queryClient.getQueryData(DASHBOARD_KEYS.currentGoal)

      // Actualizar optimísticamente
      queryClient.setQueryData(DASHBOARD_KEYS.currentGoal, newGoal)

      return { previousGoal }
    },
    // Rollback en error
    onError: (err, newGoal, context) => {
      queryClient.setQueryData(
        DASHBOARD_KEYS.currentGoal,
        context?.previousGoal
      )
    },
    // Refetch en éxito
    onSettled: () => {
      queryClient.invalidateQueries({ queryKey:
        DASHBOARD_KEYS.currentGoal })
    },
  })
}

```

3. Optimización de Imágenes

```

// Usar CDN con parámetros de consulta para optimización
const getOptimizedImageUrl = (url: string, width: number) => {
  return `${url}?w=${width}&q=80&fm=webp`
}

```

```
// En el componente PromotionCard
<img
  src={getOptimizedImageUrl(promotion.imageUrl, 800)}
  srcSet={`
    ${getOptimizedImageUrl(promotion.imageUrl, 400)} 400w,
    ${getOptimizedImageUrl(promotion.imageUrl, 800)} 800w
  `}
  sizes="(max-width: 640px) 280px, 400px"
  alt={promotion.title}
  loading="lazy"
/>
```



Actualizaciones en Tiempo Real (Mejora Opcional)

Integración WebSocket

```
// src/core/api/websocket.ts

import { io, Socket } from 'socket.io-client'
import { useQueryClient } from '@tanstack/react-query'
import { DASHBOARD_KEYS } from '@core/hooks/useDashboard'

let socket: Socket | null = null

export const initWebSocket = (token: string) => {
  socket = io(import.meta.env.VITE_WS_URL, {
    auth: { token },
    transports: ['websocket'],
  })

  return socket
}

export const useDashboardWebSocket = () => {
  const queryClient = useQueryClient()

  React.useEffect(() => {
    if (!socket) return

    // Escuchar actualizaciones de metas
    socket.on('goal:updated', (data) => {
      queryClient.setQueryData(DASHBOARD_KEYS.currentGoal, data)
    })

    // Escuchar nuevo cliente que requiere atención
    socket.on('client:attention', (data) => {
      queryClient.invalidateQueries({ queryKey:
        DASHBOARD_KEYS.clientsAttention })
    })
  }, [socket])
}
```

```

    toast.info(`Nuevo cliente requiere atención:
    ${data.clientName}`)
  })

  // Escuchar actualizaciones de comisiones
  socket.on('commission:earned', (data) => {
    queryClient.invalidateQueries({ queryKey:
      DASHBOARD_KEYS.currentGoal })
    toast.success(`¡Nueva comisión de
    ${formatCurrency(data.amount)}!`)
  })

  return () => {
    socket?.off('goal:updated')
    socket?.off('client:attention')
    socket?.off('commission:earned')
  }
}, [queryClient])
}

```

Eventos del Backend a Emitir: - goal:updated - Cuando cambia el progreso de la meta - client:attention - Nuevo cliente necesita atención - commission:earned - Nueva comisión registrada - promotion:new - Nueva promoción activada - level:upgraded - Nivel de usuario cambió

Estrategias de Testing

1. Mock de Respuestas API

```

// src/mocks/handlers.ts (MSW)

import { rest } from 'msw'

export const dashboardHandlers = [
  rest.get('/api/v1/user/profile', (req, res, ctx) => {
    return res(
      ctx.status(200),
      ctx.json({
        success: true,
        data: {
          userId: 'usr_test123',
          currentLevel: 'starter',
          accumulatedSales: 4020000,
          levelProgress: {
            currentSales: 4020000,
            nextLevelRequired: 4500000,
            percentageComplete: 89.3,
          },
        },
        commissionRate: 2.0,
      })
    )
  })
]

```

```

    },
  })
)
}),
],

rest.get('/api/v1/goals/current', (req, res, ctx) => {
  return res(
    ctx.status(200),
    ctx.json({
      success: true,
      data: {
        id: 'goal_test',
        amount: 25000,
        currentAmount: 20000,
        month: 'Octubre',
        year: 2025,
        achieved: false,
        progress: 80.0,
        remaining: 5000,
        earnedCommissions: 20000,
      },
    })
  )
}),
]

```

2. Tests de Integración

// src/pages/Dashboard.test.tsx

```

import { render, screen, waitFor } from '@testing-library/react'
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
import Dashboard from './Dashboard'
import { server } from '@mocks/server'

describe('Dashboard', () => {
  const queryClient = new QueryClient({
    defaultOptions: { queries: { retry: false } },
  })

  const wrapper = ({ children }: { children: React.ReactNode }) => (
    <QueryClientProvider client={queryClient}>{children}</QueryClientProvider>
  )

  beforeAll(() => server.listen())
  afterEach(() => server.resetHandlers())
  afterAll(() => server.close())

  it('carga y muestra datos de perfil de usuario', async () => {

```



```
render(<Dashboard />, { wrapper })





await waitFor(() => {
  expect(screen.getByText('Starter')).toBeInTheDocument()
})

it('muestra progreso de meta actual', async () => {
  render(<Dashboard />, { wrapper })





  await waitFor(() => {
    expect(screen.getByText('$25,000')).toBeInTheDocument()
    expect(screen.getByText('$20,000')).toBeInTheDocument()
  })
})
})
```

Ruta de Migración





Fase 1: Configuración (Semana 1)

1.  Crear capa de servicio API (src/core/api/dashboard.ts)
2.  Crear hooks de React Query (src/core/hooks/useDashboard.ts)
3.  Agregar MSW para desarrollo/testing
4.  Actualizar variables de entorno

Fase 2: Integración (Semana 2)

1.  Actualizar Dashboard.tsx para usar hooks
2.  Reemplazar constantes STUB_ con llamadas API
3.  Agregar estados de carga/error
4.  Implementar manejo de errores

Fase 3: Optimización (Semana 3)

1.  Agregar actualizaciones optimistas
2.  Implementar estrategias de caché
3.  Agregar polling para actualizaciones de clientes
4.  Optimizar carga de imágenes

Fase 4: Mejoras (Semana 4)

1. ☐ Agregar WebSocket para actualizaciones en tiempo real
 2. ☐ Implementar soporte offline (service worker)
 3. ☐ Agregar seguimiento de analytics
 4. ☐ Monitoreo de rendimiento
-

Recursos Adicionales

Variables de Entorno

```
# .env.local
VITE_API_BASE_URL=https://api.pazz.com
VITE_WS_URL=wss://ws.pazz.com
VITE_CDN_URL=https://cdn.pazz.com
```

Documentación API Backend

Una vez implementado el backend, la documentación completa de la API debe estar disponible en: - OpenAPI/Swagger: <https://api.pazz.com/docs> - Colección Postman: Enlace por proporcionar

Consideraciones de Seguridad

1. **Almacenamiento de Token:** Guardar en cookies httpOnly (no localStorage)
 2. **CORS:** Configurar headers CORS apropiados en backend
 3. **Rate Limiting:** Implementar en backend (100 req/min por usuario)
 4. **Validación de Input:** Tanto cliente como servidor
 5. **Inyección SQL:** Usar consultas parametrizadas en backend
 6. **Prevención XSS:** Sanitizar todas las entradas de usuario
-

Soporte y Contactos

Líder Equipo Backend: [Por asignar] **Documentación API:** [Por proporcionar] **Canal Slack:** #pazz-backend-integration

Estado del Documento: ☒ Completo - Listo para Implementación Backend **Próxima Revisión:** 2025-11-01 **Responsable:** Equipo Frontend