

Contexto de Desarrollo para IAs - Proyecto MiDinero

> Este documento proporciona contexto e instrucciones para IAs (Cursor, ChatGPT, Claude, etc.) sobre cómo trabajar efectivamente con el proyecto MiDinero.

Identidad del Proyecto

****Nombre****: MiDinero

****Tipo****: Aplicación web de control de gastos personales con IA

****Audiencia****: Gente del común, usuarios cotidianos

****Tono****: Casual, amigable, cercano (NO formal)

Stack Tecnológico

Frontend

- React 18 + TypeScript
- Vite (build tool)
- Tailwind CSS v4.0
- Shadcn/ui (componentes en `/components/ui/`)
- Motion/React (animaciones - NO Framer Motion)
- Recharts (gráficos)
- Lucide React (iconos)
- React Hook Form 7.55.0 + Zod (validación)
- Sonner 2.0.3 (toasts)

Backend (Arquitectura Híbrida)

1. ****Supabase****: Autenticación + PostgreSQL + Storage
2. ****Kotlin + Spring Boot**** (Puerto 8080): API REST para gastos y categorías
3. ****Python + FastAPI**** (Puerto 8000): Microservicio de IA para categorización y análisis

Base de Datos

PostgreSQL vía Supabase con tablas:

- `users` (Supabase Auth)
- `expenses` (id, user_id, amount, category, description, date, created_at)
- `categories` (id, name, icon, color, user_id)
- `user_preferences` (id, user_id, currency, language, theme)

Estructura de Archivos Clave

...

App.tsx → Componente raíz

components/

auth/

login-form.tsx → Autenticación

register-form.tsx

dashboard/

expense-form.tsx → Formulario principal de gastos

expense-list.tsx → Lista de gastos

expense-summary.tsx → Resumen con gráficos

ai-insights.tsx → Insights generados por IA

currency-settings.tsx → Configuración de moneda

presentation/

presentation-mode.tsx → Modo presentación interactiva
theme-provider.tsx → Provider de tema oscuro
theme-toggle.tsx → Toggle de tema
ui/ → Shadcn components (NO MODIFICAR)
utils/
hybrid-api.tsx → ****USAR ESTE**** - API híbrida principal
api-kotlin.tsx → Cliente directo de Kotlin
api.tsx → Deprecated
mock-auth.tsx → Auth mock para testing
supabase/
client.tsx → Cliente de Supabase
info.tsx → Info de configuración
styles/
globals.css → Variables CSS, tema, tipografía base
...

Reglas Críticas de Código

1. Tipografía (MUY IMPORTANTE)

****NUNCA** uses estas clases de Tailwind a menos que el usuario lo pida explícitamente:

- **✗** ``text-sm`, `text-lg`, `text-2xl`, etc. (font-size)`
- **✗** ``font-bold`, `font-semibold`, `font-light`, etc. (font-weight)`
- **✗** ``leading-tight`, `leading-normal`, etc. (line-height)`

****Razón****: La tipografía está definida en ``styles/globals.css`` para cada elemento HTML.

2. Componentes Shadcn

- ****SIEMPRE importar desde****: `./components/ui/[componente]`
- ****NO crear versiones propias**** de componentes que ya existen
- ****NO modificar archivos**** en `./components/ui/` directamente sin razón
- ****Componentes disponibles****: Button, Card, Dialog, Sheet, Tabs, Form, Input, Select, Checkbox, Switch, Alert, Toast, Skeleton, Table, Avatar, Badge, Progress, Chart, y 30+ más

3. APIs y Datos

- ****Usar `hybrid-api.tsx`**** como interfaz principal
- ****NO hardcodear**** URLs de API (usar variables de entorno)
- ****Siempre incluir**** auth token de Supabase en headers
- ****Manejar errores**** con try/catch y mostrar toasts (Sonner)
- ****Loading states**** en todas las operaciones async

4. Imports Específicos

``typescript

// Correcto

import { toast } from 'sonner@2.0.3';

import { useForm } from 'react-hook-form@7.55.0';

import { motion } from 'motion/react';

import { Button } from './components/ui/button';

// Incorrecto

import { toast } from 'sonner'; // ❌ falta versión

import { motion } from 'framer-motion'; // ❌ usar motion/react

...

5. Naming Conventions

- ****Componentes****: PascalCase (`ExpenseForm.tsx`)
- ****Utilidades****: kebab-case (`hybrid-api.tsx`)

- **Hooks personalizados**: camelCase con prefijo `use` (`useExpenses.ts`)
- **Constantes**: UPPER_SNAKE_CASE (`const API_URL = ...`)

Sistema de Diseño

Paleta de Colores (Tema Oscuro Verde)

```css

```
/* Definidas en styles/globals.css */

--primary: #10b981; /* emerald-500 - verde dinero */
--primary-dark: #059669; /* emerald-600 */
--primary-light: #34d399; /* emerald-400 */
--background: #0a0a0a; /* negro suave */
--surface: #111111; /* superficie elevada */
--card: #1a1a1a; /* cards */
--foreground: #ffffff; /* texto principal */
--muted: #737373; /* texto secundario */
...
```

### ### Uso de Colores

- **Acciones principales**: usar `#10b981` (verde)
- **Fondos**: `bg-[#0a0a0a]`, `bg-[#111111]`, `bg-[#1a1a1a]`
- **Bordes**: `border-[#10b981]` o `border-gray-800`
- **Texto**: Sin clases (usar defaults de globals.css) o `text-white`, `text-gray-400`

### ### Responsive

- **Mobile-first**: Diseñar primero para móvil
- **Breakpoints**: `sm`, `md`, `lg`, `xl` de Tailwind

- **Grids**: Usar `grid-cols-1 md:grid-cols-2 lg:grid-cols-3`

---

## ## Tono y Mensajes de la App

### ### Principio Fundamental

MiDinero está dirigido a **gente del común**, no a expertos financieros. El tono debe ser **casual, cercano y amigable**.

### ### Ejemplos de Mensajes

#### #### ❌ NO Hacer (Formal/Corporativo)

``typescript

"Se ha detectado un incremento del 23% en la categoría alimentación durante el período fiscal"

"Por favor, ingrese los datos requeridos en el formulario"

"Error: La operación no pudo ser completada exitosamente"

"Sus estadísticas financieras muestran una tendencia ascendente"

...

#### #### ✅ Sí Hacer (Casual/Amigable)

``typescript

"¡Ey! Gastaste bastante en comida esta semana 😊"

"¿En qué gastaste?"

"Uy, algo salió mal. Intenta de nuevo 🙌"

"¡Vas por buen camino! Sigues ahorrando 💪"

...

### ### Categorías de Mensajes

**\*\*Insights de IA:\*\***

```
``typescript
```

"💡 Podrías ahorrar \$250 si reduces un café por día"

"🇮🇹 Este mes gastaste menos que el anterior. ¡Bien ahí! 🎉"

"⚠️ Ojo, ya llevas \$800 en salidas. Tal vez modera un poco 😬"

"🎯 Si sigues así, ahorrarás \$500 este mes"

```
...
```

**\*\*Validaciones y Errores:\*\***

```
``typescript
```

"Ey, necesitas poner cuánto gastaste"

"El monto debe ser mayor a \$0 (¡no regalamos dinero! 😊)"

"Escribe algo sobre el gasto para recordar después"

```
...
```

**\*\*Confirmaciones y Éxito:\*\***

```
``typescript
```

"¡Listo! Gasto agregado 💚"

"¡Guardado! Ahora relájate un rato 😌"

"¡Categoría creada! 🧠"

```
...
```

**\*\*Errores del Sistema:\*\***

```
``typescript
```



















"Uy, no pudimos guardar eso 😅 Intenta de nuevo"

"Algo falló. ¿Tienes internet? 🤖"

"Oops, hubo un error. Dale otra vez"

...

### ### Uso de Emojis

-  **\*\*Sí usar\*\*** ocasionalmente para dar personalidad
-  Emojis relacionados con dinero:       
-  Emojis de reacciones:       
-  **\*\*No abusar\*\*** - máximo 1-2 por mensaje

---

## ## Endpoints de API

### ### Kotlin Backend (localhost:8080)

``typescript

// Gastos

GET /api/expenses // Lista todos los gastos del usuario  
POST /api/expenses // Crea nuevo gasto  
PUT /api/expenses/{id} // Actualiza gasto  
DELETE /api/expenses/{id} // Elimina gasto

// Categorías

GET /api/categories // Lista categorías del usuario  
POST /api/categories // Crea categoría  
PUT /api/categories/{id} // Actualiza categoría  
DELETE /api/categories/{id} // Elimina categoría

// Headers obligatorios

Authorization: Bearer {supabase\_access\_token}



Content-Type: application/json

...

### Python IA Backend (localhost:8000)

```typescript

// Categorización automática

POST /api/ai/categorize

Body: { description: string, amount: number }

Response: { category: string, confidence: number, reasoning: string }

// Análisis de patrones

POST /api/ai/analyze

Body: { expenses: Expense[], period: string }

Response: {

trends: Trend[],

insights: string[], // Mensajes casuales

suggestions: string[] // Sugerencias de ahorro

}

// Predicción de gastos futuros

POST /api/ai/predict

Body: { userId: string, category: string, months: number }

Response: {

predicted_amount: number,

confidence: number,

next_month: string

}

```
// Headers obligatorios
Authorization: Bearer {supabase_access_token}
Content-Type: application/json
...
```

```
### Supabase
```

```
``typescript
// Auth
supabase.auth.signUp({ email, password })
supabase.auth.signInWithPassword({ email, password })
supabase.auth.signOut()
supabase.auth.getSession()
```

```
// Database
supabase.from('expenses')
  .select('*')
  .eq('user_id', userId)
  .order('date', { ascending: false })
...
```

```
---
```

```
## Patrones de Código Comunes
```

```
### 1. Crear Componente con Formulario
```

```
``typescript
import { useState } from 'react';
```

```

import { useForm } from 'react-hook-form@7.55.0';
import { z } from 'zod';
import { zodResolver } from '@hookform/resolvers/zod';
import { Button } from './components/ui/button';
import { Input } from './components/ui/input';
import { Form, FormField, FormItem, FormLabel, FormControl, FormMessage } from
'./components/ui/form';
import { toast } from 'sonner@2.0.3';
import { hybridAPI } from '../utils/hybrid-api';

const formSchema = z.object({
  description: z.string().min(1, 'Ey, describe el gasto'),
  amount: z.number().positive('El monto debe ser mayor a $0')
});

export function ExpenseForm() {
  const [loading, setLoading] = useState(false);

  const form = useForm({
    resolver: zodResolver(formSchema),
    defaultValues: { description: '', amount: 0 }
  });

  const onSubmit = async (data: z.infer<typeof formSchema>) => {
    setLoading(true);
    try {
      await hybridAPI.createExpense(data);
      toast.success('¡Listo! Gasto agregado 🍀');
      form.reset();
    }
  }
}

```

```

    } catch (error) {
      toast.error('Uy, algo salió mal 😊');
    } finally {
      setLoading(false);
    }
  };

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">
        {/* campos del formulario */}
      </form>
    </Form>
  );
}
...

```

2. Llamada a API Híbrida

```

``typescript
import { hybridAPI } from '../utils/hybrid-api';
import { toast } from 'sonner@2.0.3';

async function fetchExpenses() {
  try {
    const expenses = await hybridAPI.getExpenses();
    return expenses;
  } catch (error) {
    toast.error('No pudimos cargar los gastos 😞');
  }
}

```

```
    return [];  
  }  
}
```

// Con categorización de IA

```
async function addExpenseWithAI(description: string, amount: number) {  
  try {  
    // 1. Categorizar con IA  
    const aiResult = await hybridAPI.categorizeExpense({ description, amount });  
  
    // 2. Guardar gasto  
    const expense = await hybridAPI.createExpense({  
      description,  
      amount,  
      category: aiResult.category,  
      date: new Date().toISOString()  
    });  
  
    toast.success('¡Gasto agregado y categorizado! ✨');  
    return expense;  
  } catch (error) {  
    toast.error('Algo falló al guardar 😞');  
    throw error;  
  }  
}  
...  

```

3. Componente con Gráfico (Recharts)

```
``typescript
```

```
import { BarChart, Bar, XAxis, YAxis, Tooltip, ResponsiveContainer } from 'recharts';
```

```
import { Card } from './components/ui/card';
```

```
interface ChartData {
```

```
  category: string;
```

```
  amount: number;
```

```
}
```

```
export function ExpenseChart({ data }: { data: ChartData[] }) {
```

```
  return (
```

```
    <Card className="p-6">
```

```
      <h3 className="mb-4">Gastos por Categoría</h3>
```

```
      <ResponsiveContainer width="100%" height={300}>
```

```
        <BarChart data={data}>
```

```
          <XAxis
```

```
            dataKey="category"
```

```
            stroke="#737373"
```

```
            style={{ fontSize: '12px' }}
```

```
          />
```

```
          <YAxis stroke="#737373" />
```

```
          <Tooltip
```

```
            contentStyle={{
```

```
              backgroundColor: '#1a1a1a',
```

```
              border: '1px solid #10b981',
```

```
              borderRadius: '8px'
```

```
            }}
```

```
            labelStyle={{ color: 'ffffff' }}
```

```
          />
```

```

      <Bar
        dataKey="amount"
        fill="#10b981"
        radius={[8, 8, 0, 0]}
      />
    </BarChart>
  </ResponsiveContainer>
</Card>
);
}
...

```

4. Animaciones con Motion

```

``typescript
import { motion } from 'motion/react';
import { Card } from './components/ui/card';

export function AnimatedCard({ children }: { children: React.ReactNode }) {
  return (
    <motion.div
      initial={{ opacity: 0, y: 20 }}
      animate={{ opacity: 1, y: 0 }}
      transition={{ duration: 0.3 }}
    >
      <Card className="p-6">
        {children}
      </Card>
    </motion.div>
  );
}

```

```

    );
  }

  // Lista con delay escalonado
  export function ExpenseList({ expenses }) {
    return (
      <div className="space-y-2">
        {expenses.map((expense, index) => (
          <motion.div
            key={expense.id}
            initial={{ opacity: 0, x: -20 }}
            animate={{ opacity: 1, x: 0 }}
            transition={{ delay: index * 0.1, duration: 0.3 }}
          >
            <ExpenseItem expense={expense} />
          </motion.div>
        ))}
      </div>
    );
  }
  ...
  ---

```

Instrucciones para Modificaciones

Al Agregar Nueva Feature

1. ****Identificar componentes necesarios****

- ¿Dónde va? (`/components/dashboard/`, `/components/auth/`, etc.)
- ¿Qué componentes UI usa? (Button, Card, Form, etc.)
- ¿Necesita API? Usar `hybrid-api.tsx`

2. ****Mantener consistencia****

- Usar mismo patrón de imports
- Seguir naming conventions
- Aplicar tema verde oscuro
- Tono casual en mensajes

3. ****Siempre incluir****

- Loading states (`useState<boolean>`)
- Error handling (try/catch + toast)
- TypeScript types explícitos
- Responsive design

4. ****Actualizar archivos relacionados****

- Si creas nueva ruta, actualizar `App.tsx`
- Si agregas endpoint, documentar en hybrid-api
- Si creas tipo, agregarlo a `/src/types/`

Al Refactorizar Código

1. ****Preservar funcionalidad existente****

- No cambiar comportamiento sin indicación
- Mantener compatibilidad con API
- No romper imports existentes

2. ****Mejorar estructura****

- Extraer lógica a custom hooks si es complejo
- Separar concerns (UI vs lógica)
- Reducir duplicación

3. ****Mantener estilos****

- No cambiar tema verde oscuro
- Preservar responsive behavior
- No agregar clases de tipografía innecesarias

Al Debuggear

1. ****Revisar primero****

- Console errors en navegador
- Network tab para llamadas API
- Supabase logs si es auth/DB

2. ****Errores comunes****

- CORS: Backend debe permitir `http://localhost:5173`
- Auth: Verificar token en headers
- Imports: Versión correcta de paquetes
- Tailwind: Clases mal escritas o inexistentes

3. ****Logging útil****

```
````typescript
console.log('[ExpenseForm] Submitting:', data);
console.log('[API] Response:', response);
console.error('[Error]', error.message);
````
```

Casos de Uso Frecuentes

1. Agregar Nuevo Gráfico

``typescript

// 1. Crear componente en /components/dashboard/

// 2. Usar Recharts con tema verde

// 3. Datos desde hybrid-api

// 4. Card wrapper de shadcn

// 5. Responsive con ResponsiveContainer

```
import { LineChart, Line, XAxis, YAxis, Tooltip, ResponsiveContainer } from 'recharts';
```

```
export function TrendChart({ data }) {
```

```
  return (
```

```
    <Card className="p-6">
```

```
      <h3 className="mb-4">Tendencia de Gastos</h3>
```

```
      <ResponsiveContainer width="100%" height={250}>
```

```
        <LineChart data={data}>
```

```
          <XAxis dataKey="month" stroke="#737373" />
```

```
          <YAxis stroke="#737373" />
```

```
          <Tooltip
```

```
            contentStyle={{
```

```
              backgroundColor: '#1a1a1a',
```

```
              border: '1px solid #10b981'
```

```
            }}
```

```
          />
```

```

    <Line
      type="monotone"
      dataKey="amount"
      stroke="#10b981"
      strokeWidth={2}
    />
  </LineChart>
</ResponsiveContainer>
</Card>
);
}
...

```

2. Integrar Nuevo Endpoint de IA

```

``typescript
// En utils/hybrid-api.tsx

export const hybridAPI = {
  // ... métodos existentes

  // Nuevo método de IA
  async getBudgetSuggestions(userId: string) {
    const token = await getAuthToken();

    const response = await fetch(`${PYTHON_API}/api/ai/budget-suggestions`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,

```

```

    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ userId })
});

if (!response.ok) throw new Error('Failed to get suggestions');
return response.json();
}
};

// Usar en componente
import { hybridAPI } from '../utils/hybrid-api';

async function loadSuggestions() {
  try {
    const suggestions = await hybridAPI.getBudgetSuggestions(userId);
    // Mostrar con tono casual
    suggestions.forEach(s => toast.info(s.message));
  } catch (error) {
    toast.error('No pudimos generar sugerencias 🤖');
  }
}
...

```

3. Crear Sistema de Notificaciones

```

````typescript
import { toast } from 'sonner@2.0.3';

```

```
// Success con mensaje casual
toast.success('¡Categoría creada! 🎉');

// Error amigable
toast.error('Uy, algo salió mal 😞 Intenta de nuevo');

// Info con insight de IA
toast.info('💡 Tip: Podrías ahorrar $50 esta semana');

// Warning
toast.warning('⚠️ Ya llevas $500 en salidas este mes');

// Loading toast
const toastId = toast.loading('Guardando...');
// ... operación async
toast.success('¡Guardado! 💾', { id: toastId });
...

```

#### ### 4. Implementar Feature Responsive

```
``typescript
export function Dashboard() {
 return (
 <div className="container mx-auto p-4">
 {/* Header mobile/desktop */}
 <div className="flex flex-col md:flex-row md:items-center md:justify-between mb-6">
 <h1>Dashboard</h1>
 <Button>Agregar Gasto</Button>

```

```
</div>
```

```
{/* Grid responsive */}
```

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
```

```
 <ExpenseSummary />
```

```
 <CategoryChart />
```

```
 <AllInsights />
```

```
</div>
```

```
{/* Lista con sidebar en desktop */}
```

```
<div className="mt-8 grid grid-cols-1 lg:grid-cols-3 gap-6">
```

```
 <div className="lg:col-span-2">
```

```
 <ExpenseList />
```

```
 </div>
```

```
 <div className="lg:col-span-1">
```

```
 <RecentActivity />
```

```
 </div>
```

```
</div>
```

```
</div>
```

```
);
```

```
}
```

```
``
```

```

```

```
Variables de Entorno
```

```
```env
```

```
# Supabase (obligatorias)
```

VITE_SUPABASE_URL=https://tu-proyecto.supabase.co

VITE_SUPABASE_ANON_KEY=tu-anon-key

Backend URLs (opcional, defaults a localhost)

VITE_KOTLIN_API_URL=http://localhost:8080

VITE_PYTHON_API_URL=http://localhost:8000

Desarrollo

VITE_DEV_MODE=true

...

Acceso en código:

```typescript

const supabaseUrl = import.meta.env.VITE\_SUPABASE\_URL;

...

---

## Testing y Validación

### Checklist antes de considerar completa una feature:

- [ ] TypeScript compila sin errores (`tsc --noEmit`)
- [ ] Funciona en mobile, tablet, desktop
- [ ] Loading states visibles
- [ ] Errores manejan con toast casual
- [ ] Tema verde oscuro aplicado correctamente
- [ ] No usa clases de tipografía innecesarias
- [ ] Imports correctos (versiones específicas donde aplica)



- [ ] Mensajes con tono casual/amigable
- [ ] Componentes Shadcn usados (no recreados)
- [ ] API calls usan hybrid-api.tsx

---

## ## Notas Finales para IAs

### ### Prioridades al Generar Código

1. **Funcionalidad correcta** - Debe funcionar
2. **Consistencia con proyecto** - Seguir patrones existentes
3. **Tono casual** - Mensajes amigables
4. **Tema visual** - Verde oscuro siempre
5. **TypeScript estricto** - Tipos explícitos
6. **Responsive** - Mobile-first

### ### Qué NO Hacer

- ✗ Crear componentes UI personalizados si existe en Shadcn
- ✗ Usar tono formal/corporativo en mensajes
- ✗ Agregar clases `text-\*`, `font-\*`, `leading-\*` sin razón
- ✗ Ignorar manejo de errores
- ✗ Hardcodear URLs o API keys
- ✗ Usar `any` en TypeScript
- ✗ Importar paquetes sin versión cuando se requiere

### ### Qué Sí Hacer

- ✓ Usar componentes de `/components/ui/`
- ✓ Mensajes casuales con emojis ocasionales
- ✓ Mantener tema verde oscuro (#10b981)
- ✓ Try/catch + toast en operaciones async
- ✓ Variables de entorno para configuración
- ✓ Types explícitos en TypeScript
- ✓ Importar con versiones específicas cuando aplique
- ✓ Consultar hybrid-api.tsx para datos

---

## ## Documentación Adicional

Para contexto adicional, consultar:

- `CONTEXTO-PARA-OTRAS-IAs.md` - Contexto completo con ejemplos
- `MiDinero-Arquitectura-Backend-Completa.md` - Detalles de arquitectura
- `MiDinero-Proyecto-Completo-Para-ChatGPT.md` - Resumen ejecutivo
- `GUIA-PRESENTACION-INTERACTIVA.md` - Modo presentación
- `styles/globals.css` - Sistema de diseño y variables CSS

---

**\*\*Versión\*\*:** 2.0

**\*\*Última actualización\*\*:** Octubre 2025

**\*\*Generado para\*\*:** Cursor AI, ChatGPT, Claude, y otras IAs de desarrollo