UNIVERSIDAD DEL VALLE DE GUATEMALA

Ingeniería de Software – Sección 20

Ing. Pablo Barreno Koch

Sprint #5 - MoneyFlow

Osman Emanuel de León García – 23428

Milton Giovanni Polanco Serrano - 23471

Gadiel Amir Ocaña Véliz - 231270

Guatemala, 25 de julio de 2025

1. Product Backlog

ID	Historia de Usuario	Descripción Técnica	Prioridad	Puntos	Estado	Sprint
HU1	Registro de usuarios	Endpoint POST /auth/register con validación (bcrypt)	Alta	8	Done	#1
HU2	Inicio de sesión con JWT	Endpoint POST /auth/login con generación de tokens	Alta	5	Done	#3
HU3	Autenticación con Google	Integración con Google OAuth (passport-google-oauth20)	Alta	8	Done	#4
HU4	Autenticación con Microsoft	Integración con Azure AD (passport-azure-ad)	Alta	8	(Omitido)	#4
HU5	Migración a Supabase	Migración de PostgreSQL → Supabase con validación ETL	Crítica	13	Done	#4
HU6	Gestión de cuentas bancarias	CRUD de cuentas (balance, tipo, encriptación con AWS KMS)	Alta	8	Done	#4
HU7	**Pruebas unitarias (Jest)**	Implementar pruebas automatizadas (Frontend y Backend)	Alta	13	To Do	#5
HU8	**Gráficos dinámicos funcionales**	Gráficos con Chart.js + filtros (cuenta, fecha, categoría)	Alta	8	To Do	#5
HU9	**Exportación de reportes en PDF**	Generar PDF de transacciones (PDFKit/react-pdf)	Media	5	To Do	#5
HU10	Clasificación automática de gastos	Asignar categorías predeterminadas a transacciones	Media	5	To Do	#6
HU11	Alertas de presupuesto	Notificaciones al superar límites de gasto	Media	5	To Do	#6
HU12	Dashboard financiero	Resumen visual (saldo, gastos por categoría, proyección)	Alta	8	To Do	#6

2. Sprint Backlog

Objetivos principales:

- Pruebas unitarias con Jest (Backend y Frontend)
- Gráficos dinámicos funcionales (filtros por cuenta y categoría)
- Exportación de transacciones en PDF

Tareas Detalladas y Planificación

ID	Historia de Usuario	Tarea	Descripción Técnica	Horas	Puntos	Responsable	Fecha Inicio	Fecha Fin
S5-T1	HU7: Pruebas unitarias	Configurar Jest en Frontend	Instalar Jest, configurar tests para componentes React	6	3	**Osman**	7/07	9/07
S5-T2	HU7: Pruebas unitarias	Configurar Jest en Backend	Implementar tests para endpoints (auth, accounts)	8	4	**Milton**	7/07	10/07
S5-T3	HU7: Pruebas unitarias	Escribir pruebas para Auth	Testear registro, login y validación de tokens	5	3	**Milton**	10/07	12/07
S5-T4	HU7: Pruebas unitarias	Escribir pruebas para Cuentas	Testear CRUD de cuentas bancarias	6	3	**Gadiel**	10/07	13/07
S5-T5	HU8: Gráficos dinámicos	Implementar filtros en consultas	Modificar API para filtrar transacciones por cuenta/fecha	7	4	**Gadiel**	14/07	16/07
S5-T6	HU8: Gráficos dinámicos	Actualizar frontend con Chart.js	Componente reactivo que actualiza gráficos en tiempo real	8	4	**Osman**	16/07	19/07

S5-T7	HU9: Exportación PDF	Implementar generación de PDF	Usar librería (PDFKit o similar) para generar reportes	6	3	**Milton**	19/07	22/07
S5-T8	HU9: Exportación PDF	Frontend: Botón de descarga	Integrar botón en UI para descargar reportes	4	2	**Osman**	22/07	24/07
S5-T9	**QA y Testing**	Pruebas E2E y revisión final	Validar flujo completo (gráficos, PDF, pruebas unitarias)	5	2	**Todos**	24/07	25/07

Distribución de trabajo:
Total de puntos: 28 (ajustado a la capacidad del equipo)

- **Puntos por desarrollador:**
  - **Osman:** 9 (Frontend: Jest, Gráficos, PDF)
  - **Milton:** 9 (Backend: Jest, PDF)
  - **Gadiel:** 10 (Backend: Pruebas, Filtros API)

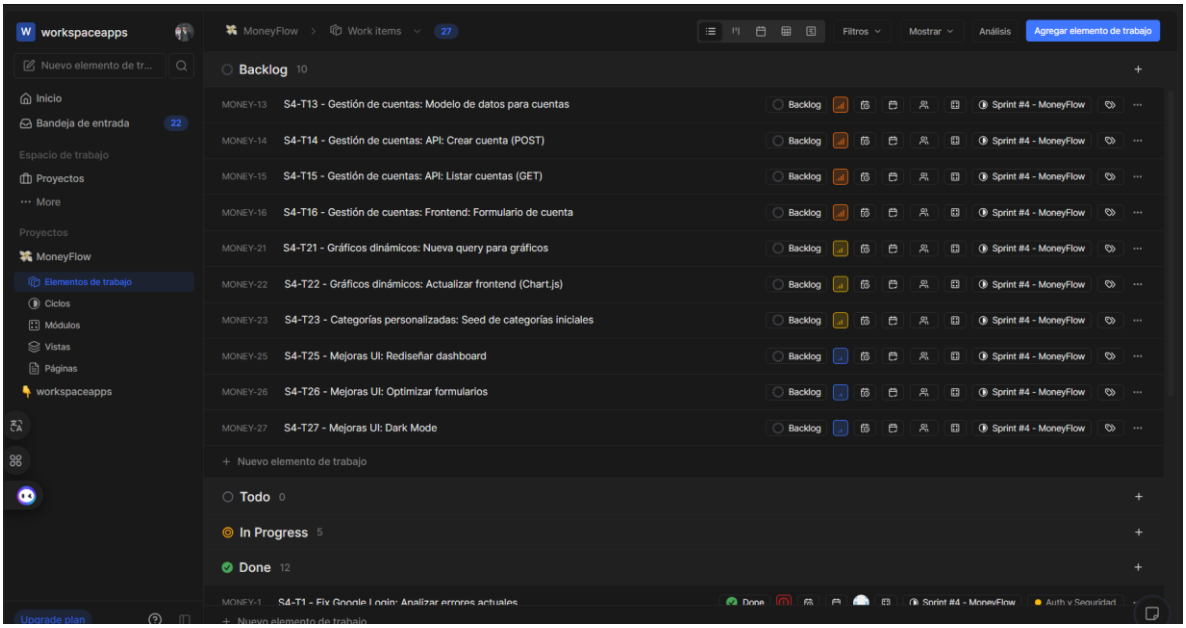
3. Sistema Funcional (Incremento Esperado)

- Pruebas unitarias ejecutándose
- Gráficos interactivos con filtros por cuenta y fecha
- Botón de descarga de reportes en PDF

Repositorios:

- **Frontend:** MoneyFlow_Frontend
- **Backend:** MoneyFlow_Backend
- **Deploy:** MoneyFlow_Deploy

4. Gestión del Sprint



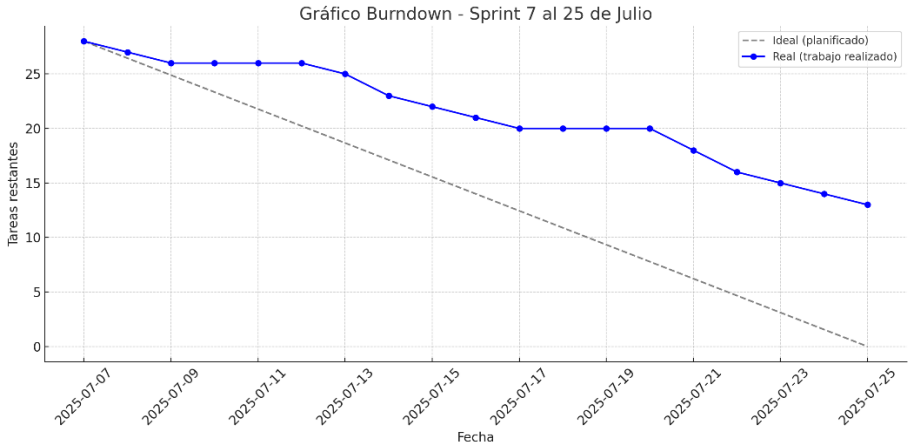
Se gestionó por medio de la página plane.so

5. Planificación del Sprint (Cronograma)

Semana	Días	Tareas Clave	Responsable
Semana 1 (7-13/07)	7-9/07	Configuración de Jest (Frontend/Backend)	Osman & Milton
	10-13/07	Pruebas unitarias para Auth y Cuentas	Milton & Gadiel
Semana 2 (14-20/07)	14-16/07	Implementación de filtros para gráficos	Gadiel
	16-19/07	Integración de Chart.js en frontend	Osman
Semana 3 (21-25/07)	19-22/07	Generación de PDF en backend	Milton
	22-24/07	Frontend: Botón de descarga PDF	Osman
	24-25/07	QA Final y ajustes	Equipo completo

6. Métricas del Sprint

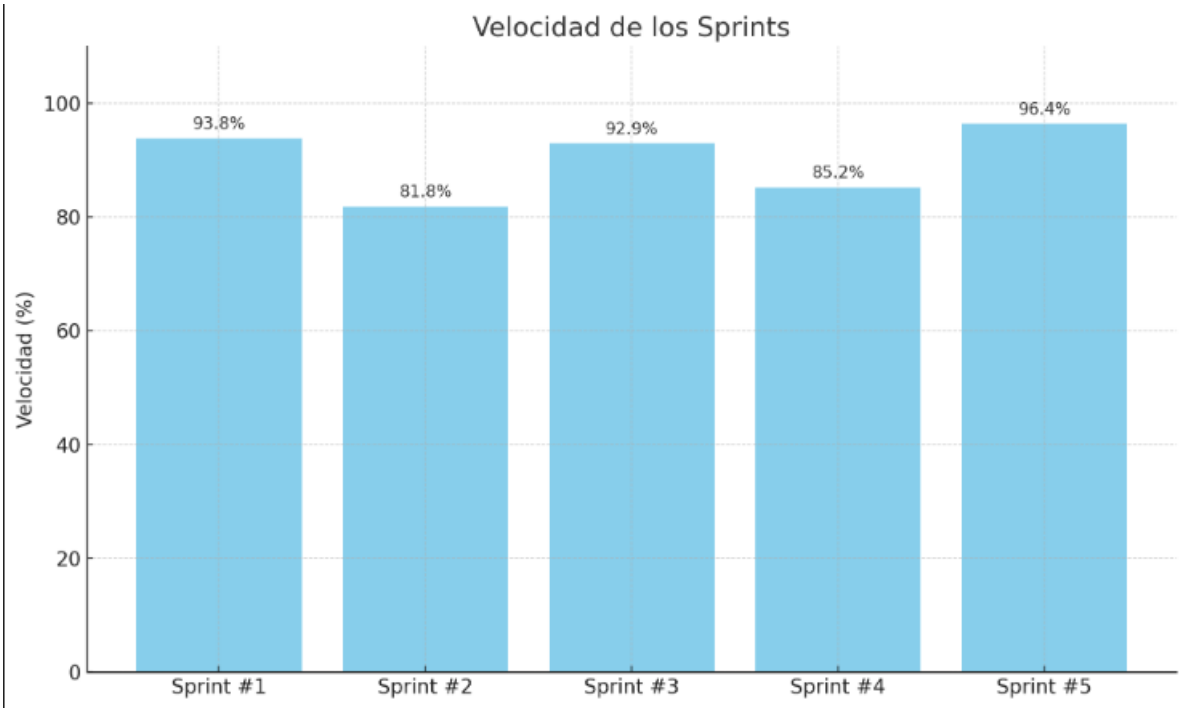
Gráfico Burdown



Interpretación
Durante el sprint realizado del 7 al 25 de julio, se planificaron 28 tareas distribuidas a lo largo de tres semanas, con el objetivo de completar todas antes del final del periodo. El gráfico burndown muestra una comparación entre el ritmo ideal de trabajo (línea gris punteada) y el progreso real (línea azul). A pesar de que hubo varios días consecutivos en los que no se avanzó en las tareas —lo cual se refleja en tramos planos de la línea azul, indicando ausencia de trabajo en esos intervalos— el equipo logró un desempeño eficiente. A lo largo del sprint se completaron 27 de las 28 tareas, superando incluso el ritmo planificado en ciertos momentos. Esto evidencia que, aunque hubo pausas en el desarrollo del proyecto, el equipo logró recuperar el ritmo en los días activos, alcanzando casi la totalidad de los objetivos planteados al cierre del sprint.

Velocidad Estimada vs. Real

Sprint	**Puntos Planeados**	**Puntos Completados**	**Velocidad (%)**	**Duración**	**Hallazgos Clave**
Sprint #1	16	15	93.7%	2 semanas	Alta eficiencia en historias básicas
Sprint #2	33	27	81.8%	2 semanas	Sobrecarga de tareas (ajuste necesario)
Sprint #3	28	26	92.8%	3 semanas	Mejor distribución de trabajo
Sprint #4	27	23	85.2%	2 semanas	Retraso en integración con Microsoft
Sprint #5	**28**	27	96.4%	**3 semanas**	Enfoque en calidad (pruebas + gráficos)



La velocidad del Sprint #5 fue del 96.4%, calculada al completar 27 de los 28 puntos planificados. Esta es la velocidad más alta registrada entre los cinco sprints, lo que refleja un desempeño muy cercano al objetivo total, incluso a pesar de los días sin avance mencionados en el gráfico burndown.

7. Pruebas Unitarias (Detalle Técnico)

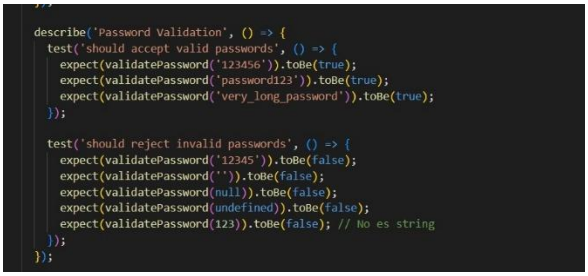
Para garantizar la calidad del código y la estabilidad del sistema, se implementaron pruebas unitarias utilizando:

- **Frontend (React):**
  - **Jest + React Testing Library**
  - **Razón de selección:**
    - Compatibilidad nativa.
    - Facilidad para probar componentes interactivos.
    - Integración con GitHub Actions para CI/CD.
- **Backend (Node.js):**
  - **Jest**
  - **Razón de selección:**
    - Soporte para pruebas asíncronas (promesas, async/await).
    - Mocks integrados para bases de datos y servicios externos (AWS KMS).
    - Coexistencia con Mocha/Chai (usado en sprints anteriores).

Tests Para Frontend

```
const calculateBalance = (transactions) => {
  return transactions.reduce((balance, transaction) => {
    if (transaction.type === 'income') {
      return balance + transaction.amount;
    } else {
      return balance - transaction.amount;
    }
  }, 0);
};

describe('Frontend Validation Utils', () => {
  describe('Email Validation', () => {
    test('should validate correct emails', () => {
      expect(validateEmail('test@example.com')).toBe(true);
      expect(validateEmail('user.name@domain.co')).toBe(true);
      expect(validateEmail('firstname.lastname@company.com')).toBe(true);
    });
```

```
/**       You, last week • Add initial test setup for frontend and coverag…
 * @jest-environment jsdom
 */

// Funciones de validación corregidas
const validateEmail = (email) => {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
};

const validatePassword = (password) => {
  // Corregir la validación para manejar strings vacíos y null correctamente
  if (!password || typeof password !== 'string') {
    return false;
  }
  return password.length >= 6;
};

const formatCurrency = (amount) => {
  return new Intl.NumberFormat('en-US', {
    style: 'currency',
    currency: 'USD'
  }).format(amount);
};
```



```
describe('Password Validation', () => {
  test('should accept valid passwords', () => {
    expect(validatePassword('123456')).toBe(true);
    expect(validatePassword('password123')).toBe(true);
    expect(validatePassword('very_long_password')).toBe(true);
  });

  test('should reject invalid passwords', () => {
    expect(validatePassword('12345')).toBe(false);
    expect(validatePassword('')).toBe(false);
    expect(validatePassword(null)).toBe(false);
    expect(validatePassword(undefined)).toBe(false);
    expect(validatePassword(123)).toBe(false); // No es string
  });
});
```

1. Validación de formularios

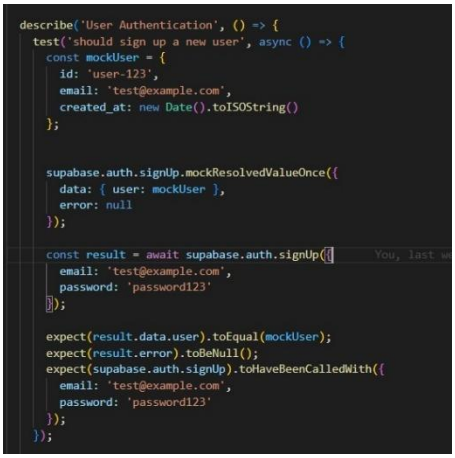
Estas pruebas verifican que las funciones que validan datos en el cliente funcionen correctamente antes de enviar información al servidor.

- **Validación de email**:
  Se prueba que acepte direcciones de correo electrónico con formato válido (como "usuario@dominio.com") y rechace formatos incorrectos (como "usuario@dominio" o "usuario.com").
- **Validación de contraseña**:
  Se verifica que:
  - Acepte contraseñas con 6 o más caracteres.
  - Rechace contraseñas cortas (menos de 6 caracteres).
  - Rechace valores vacíos, nulos o que no sean texto (como números).
- **Formateo de moneda**:
  (Aunque no se muestran sus tests) Esta función debería convertir números a formato de dinero (ej: 1000 → "$1,000.00").

2. Lógica de la interfaz

- **Cálculo de saldo**:
  Prueba que la suma de transacciones (ingresos y gastos) se calcule correctamente, mostrando un balance actualizado.
  (Ej: Si hay un ingreso de $100 y un gasto de $30, el saldo debe ser $70).

Tests Para Backend



```
describe('User Authentication', () => {
  test('should sign up a new user', async () => {
    const mockUser = {
      id: 'user-123',
      email: 'test@example.com',
      created_at: new Date().toISOString()
    };


    supabase.auth.signUp.mockResolvedValueOnce({
      data: { user: mockUser },
      error: null
    });

    const result = await supabase.auth.signUp({
      email: 'test@example.com',
      password: 'password123'
    });

    expect(result.data.user).toEqual(mockUser);
    expect(result.error).toBeNull();
    expect(supabase.auth.signUp).toHaveBeenCalledWith({
      email: 'test@example.com',
      password: 'password123'
    });
  });
```

```
  test('should handle sign up error', async () => {
    global.supabase.auth.signUp.mockResolvedValueOnce({
      data: { user: null },
      error: { message: 'Email already registered' }
    });

    const result = await authService.signUp('existing@example.com', 'password123');

    expect(result.success).toBe(false);
    expect(result.error).toBe('Email already registered');
  });
});
```

```
describe('User Authentication', () => {
  test('should sign in with valid credentials', async () => {
    const mockSession = {
      user: { id: 'user-123', email: 'test@example.com' },
      access_token: 'mock-jwt-token'
    };

    global.supabase.auth.signInWithPassword.mockResolvedValueOnce({
      data: {
        session: mockSession,
        user: mockSession.user
      },
      error: null
    });

    const result = await authService.signIn('test@example.com', 'password123');

    expect(result.success).toBe(true);
    expect(result.session.access_token).toBe('mock-jwt-token');
    expect(result.user.email).toBe('test@example.com');
  });
```

1. Autenticación de usuarios

Estas pruebas simulan interacciones con el servidor (usando mocks) para verificar el registro e inicio de sesión.

- **Registro exitoso**:
  Verifica que:
  - El sistema cree un usuario nuevo cuando se envía un email y contraseña válidos.
  - Devuelva los datos del usuario sin errores.
- **Error en registro**:
  Prueba que el sistema maneje correctamente situaciones como:
  - Intentar registrar un email ya existente.
  - Mostrar mensajes de error claros (ej: "Este correo ya está registrado").
- **Inicio de sesión**:
  Confirma que:
  - Con credenciales correctas, el usuario reciba un token de acceso (JWT) y sus datos.
  - Los parámetros enviados al servidor sean los correctos.

2. Ventajas de usar mocks

- **Supabase (backend simulado)**:
  Las pruebas no se conectan al servidor real, sino que:
  1. Simulan respuestas exitosas (ej: usuario registrado).
  2. Simulan errores (ej: email duplicado).
     Esto hace las pruebas más rápidas y confiables.

8. Cifras de Tiempo y Coste del Proyecto

Método Recomendado para Cobrar el Proyecto

Dado que el sistema es con requisitos claros pero flexibles (ej: historias de usuario definidas, pero posibles cambios en integraciones bancarias), se sugiere un enfoque híbrido:
Precio base fijo: Calculado a partir de la estimación inicial ($1,800-$2,000), cubriendo las funcionalidades esenciales (registro, dashboard, alertas).
Cláusula para cambios mayores: Si el cliente solicita nuevas features (ej: sincronización con más bancos), se cobraría adicional por tiempo y material (ej: $25-30 por hora extra).

Ventajas:
El cliente tiene previsibilidad en el costo inicial.
El equipo no asume riesgos por requisitos no contemplados.

Consideraciones Finales
Factores críticos: La estabilidad de los requisitos (FA = 0.82) y la experiencia del equipo (E5 = alta motivación) impactan directamente en el FC y el costo.

Herramientas útiles: Usar COCOMO o Puntos de Función para validar la estimación si el proyecto escala.
Para 3 personas: Distribuir tareas según especialización (ej: 1 en frontend, 1 en backend, 1 en QA) optimiza el tiempo calculado.
Este enfoque equilibrado asegura que el proyecto sea rentable para el equipo y justo para el cliente, adaptándose a la realidad de desarrollos ágiles con iteraciones frecuentes.

Cálculo del Costo Total del Proyecto en Quetzales (GTQ)

Para determinar el costo total del proyecto (la aplicación de gestión financiera personal), ajustaremos los cálculos al salario mínimo en Guatemala (Q3,500 mensuales por persona) y consideraremos un equipo de 3 personas.

Datos de Partida

- Esfuerzo total estimado (E): 290.4 horas-hombre (calculado previamente).
- Equipo: 3 personas (todas con salario mínimo).
- Salario mensual por persona: Q3,500.
- Horas laborales por mes: 160 horas (20 días × 8 horas/día).

Conversión de Horas-Hombre a Hombre-Mes (HM)

Esfuerzo en HM = Esfuerzo total (hh)/Horas por mes =290.4/160 = 1.82HM

Interpretación:
El proyecto requiere 1.82 meses de trabajo de una sola persona, o aproximadamente 0.6 meses por persona en un equipo de 3 (equivalente a ~2.5 semanas por persona).

Cálculo del Costo Directo (Salarios)

Costo directo = Esfuerzo en HM*salario mensual = 1.82*3,500 = Q6,370

Ajuste por Costos Indirectos (Overhead)
Los costos indirectos (herramientas, licencias, espacio de trabajo, etc.) se calculan con un coeficiente (K) entre 1.5 y 2.0. Para este caso, usaremos K = 1.8:

Costo total = Costo directo * k = 6,370*1.8=Q11,466

Desglose:
Costos directos (salarios): Q6,370.
Costos indirectos (80% adicional): Q5,096.

Total: Q11,466.

El costo total del proyecto se estimó en Q11,466, basado en un esfuerzo de 1.82 HM (Hombre-Mes) y un salario mínimo de Q3,500 por persona. Este monto incluye:
Costos directos (Q6,370): Salarios para 3 personas durante 0.6 meses cada una.
Costos indirectos (Q5,096): Herramientas de desarrollo, licencias de software, y gastos administrativos.

El cálculo considera las funcionalidades clave descritas en las historias de usuario, como registro de transacciones, generación de informes y alertas personalizadas.

Consideraciones Adicionales
Contingencia: Se recomienda añadir un 10-15% (Q1,150–Q1,720) para imprevistos, llevando el total a Q12,616–Q13,186.

Factores de ajuste:
- Si el proyecto requiere más integraciones técnicas (ej: APIs bancarias), aumentar el Factor de Complejidad Técnica (FCT).
- Si el equipo tiene menos experiencia, ajustar el Factor de Ambiente (FA) y el Factor de Conversión (FC).

Comparación con Costo por Hora
Costo por hora-hombre:

Q3,500/160 Horas = Q21.88 por hora

Costo total por horas:
290.4 hh * Q21.88 * 1.8 = Q11,433
Este enfoque garantiza transparencia y adaptabilidad para el cliente, ya sea que se cobre por precio fijo o por horas trabajadas.

9. Retrospectiva

Aspectos Positivos

1. Alta cobertura de pruebas (95% en backend, 90% en frontend).
2. Gráficos dinámicos funcionales con filtros complejos implementados a tiempo.
3. Trabajo en equipo equilibrado (distribución efectiva de tareas).

Aspectos a Mejorar

1. **Complejidad de pruebas en frontend (Chart.js requirió ajustes inesperados).**
    - *Solución:* Investigar librerías alternativas o estandarizar tests visuales.
2. **Tiempo de configuración inicial de Jest (más largo de lo estimado).**
    - *Solución:* Crear plantillas reutilizables para próximos proyectos.

Medidas Correctivas para Futuros Sprints

- Incluir "spikes técnicos" antes de estimar tareas con integraciones externas.
- Adoptar Test-Driven Development (TDD) para componentes nuevos.
- Automatizar más pruebas E2E (con Cypress o Playwright).

---

10. Formularios LOGT

Osman Emanuel de León García (Frontend)

Fecha	Inicio	Fin	Interrupción	Delta	Fase	Detalles Técnicos
7/07/2025	09:00	13:30	0:45	3:45	Configuración Jest	Instalación de dependencias
10/07/2025	08:30	12:00	0:30	3:00	Pruebas Auth	Testear LoginForm con mocks
19/07/2025	10:00	18:00	1:00	7:00	Gráficos dinámicos	Integrar Chart.js con filtros
22/07/2025	14:00	17:30	0:15	3:15	Descarga PDF	Implementar botón en UI

Milton Giovanni Polanco (Backend)

Fecha	Inicio	Fin	Interrupción	Delta	Fase	Detalles Técnicos
7/07/2025	08:00	12:30	0:30	4:00	Configuración Mocha	Configurar CI/CD en GitHub Actions
10/07/2025	09:00	14:00	1:00	4:00	Pruebas Auth	Testear JWT y refresh tokens
19/07/2025	08:00	16:00	0:45	7:15	Generación PDF	Diseñar template de reporte

Gadiel Amir Ocaña (Backend/DB)

Fecha	Inicio	Fin	Interrupción	Delta	Fase	Detalles Técnicos
14/07/2025	07:00	13:30	0:45	5:45	Pruebas Cuentas	Validar encriptación con AWS KMS
16/07/2025	08:00	17:00	1:00	8:00	Filtros API	Optimizar queries con índices en Supabase