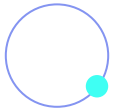
Resolución de la etapa 3

*Realizar el **modelo de negocio** en Javascript con la ayuda de ChatGPT.*

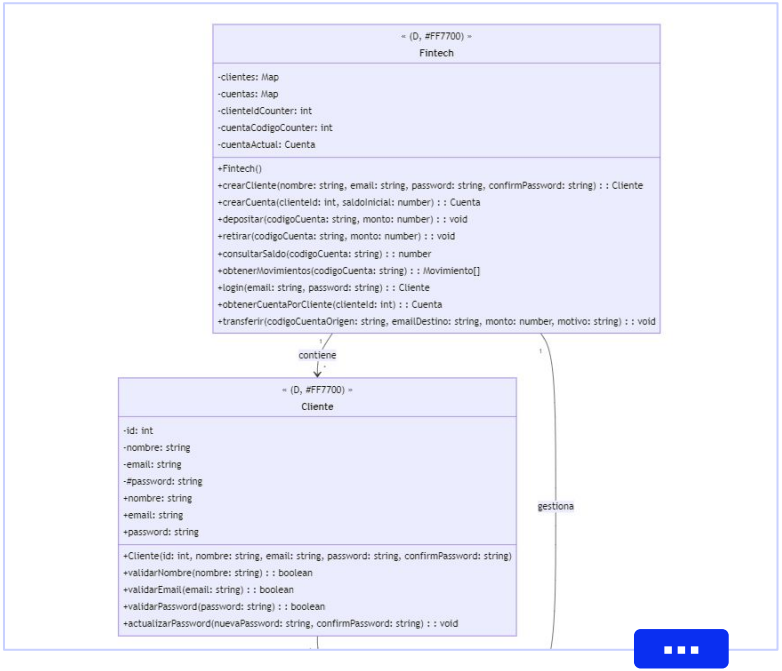
Recuerda que puedes pedirle a la IA un **diagrama Mermaid** del modelo para visualizar la solución. Utiliza la herramienta Mermaid Live Editor. Veámoslo en el siguiente *slide.*

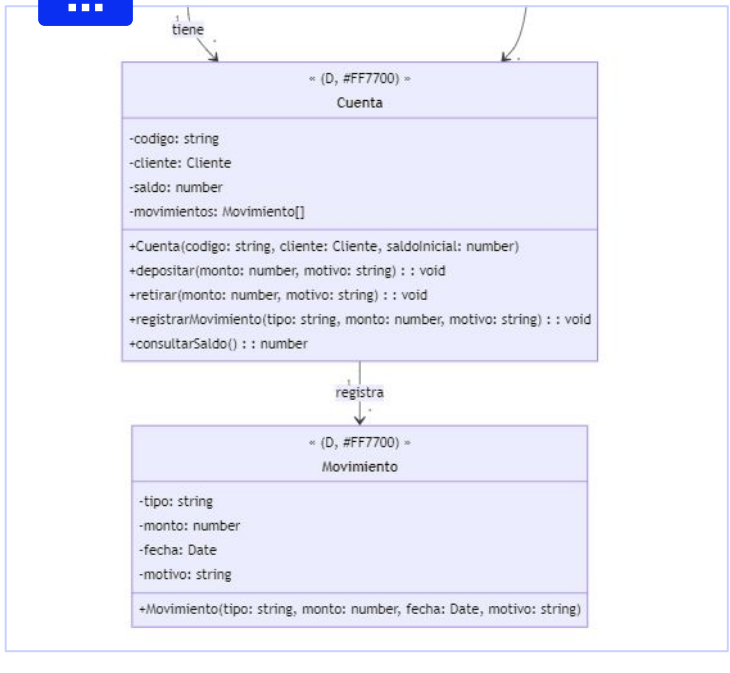


Observaciones

- Recuerda que **la primera versión de la IA no es la óptima porque no sabe las reglas de negocio o validación.**
- Debes **iterar sobre la propuesta del LLM** para pedirle qué validaciones queremos, que respete el encapsulamiento, y otros requisitos.
- Ya tienes la clase `Cliente` de la etapa anterior.

Diagrama Mermaid generado por ChatGPT





Clase *Movimiento*

```
class Movimiento {
    #tipo;
    #monto;
    #fecha;
    #motivo;

    constructor(tipo, monto, fecha, motivo) {
        // Validaciones en el constructor
        if (!this.validarTipo(tipo)) {
            throw new Error('Tipo de movimiento inválido. Debe ser "Depósito" o "Retiro".');
        }
        if (!this.validarMonto(monto)) {
            throw new Error('Monto de movimiento inválido. Debe ser mayor que cero.');
        }
        if (!this.validarFecha(fecha)) {
            throw new Error('Fecha de movimiento inválida. Debe ser una instancia de Date.');
        }
        if (!this.validarMotivo(motivo)) {
            throw new Error('Motivo de movimiento inválido. Debe ser una cadena no vacía.');
        }

        this.#tipo = tipo;
        this.#monto = monto;
        this.#fecha = fecha;
        this.#motivo = motivo;
    }
```

...

```
    // Getter para tipo
    get tipo() {
        return this.#tipo;
    }

    // Getter para monto
    get monto() {
        return this.#monto;
    }

    // Getter para fecha
    get fecha() {
        return this.#fecha;
    }

    // Getter para motivo
    get motivo() {
        return this.#motivo;
    }

    // Método para validar el tipo de movimiento ('Depósito' o 'Retiro')
    validarTipo(tipo) {
        return tipo === 'Depósito' || tipo === 'Retiro';
    }

    // Método para validar el monto (por ejemplo, debe ser mayor que cero)
    validarMonto(monto) {
        return monto > 0;
    }
```

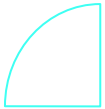
```
    // Método para validar la fecha (por ejemplo, debe ser una instancia de Date)
    validarFecha(fecha) {
        return fecha instanceof Date;
    }

    // Método para validar el motivo (por ejemplo, debe ser una cadena no vacía)
    validarMotivo(motivo) {
        return typeof motivo === 'string' && motivo.trim().length > 0;
    }
}
```

Clase *Cuenta*

```
class Cuenta {
    constructor(codigo, cliente, saldoInicial = 0) {
        this.codigo = codigo; // Código único
        this.cliente = cliente; // Referencia al objeto Cliente
        this.saldo = saldoInicial;
        this.movimientos = []; // Lista de movimientos
    }

    // Método para depositar dinero
    depositar(monto, motivo) {
        if ((monto) && (monto>0)){
            this.saldo += monto;
            this.registrarMovimiento('Depósito', monto, motivo);
        }else{
            throw new Error("Error ingresando monto vacio o negativo")
        }
    }
```

...

```
// Método para retirar dinero
    retirar(monto, motivo) {
        if (monto > this.saldo) {
            throw new Error('Saldo insuficiente');
        }
        this.saldo -= monto;
        this.registrarMovimiento('Retiro', monto, motivo);
    }

    // Método para registrar un movimiento
    registrarMovimiento(tipo, monto, motivo) {
        const movimiento = new Movimiento(tipo, monto, new Date(), motivo);
        this.movimientos.push(movimiento);
    }

    // Método para consultar el saldo
    consultarSaldo() {
        return this.saldo;
    }
}
```



Pruebas Unitarias

Crear un usuario nuevo

```
// crearUsuario.test.js
const { Fintech } = require('./path-to-your-classes'); // Ajusta el path según sea necesario

test('Crear un usuario nuevo', () => {
    const fintech = new Fintech();
    fintech.crearUsuario('usuario1', 'contraseña1');
    expect(fintech.usuarios.has('usuario1')).toBe(true);
});
```

Login exitoso

```
// loginExitoso.test.js
const { Fintech } = require('./path-to-your-classes'); // Ajusta el path según sea necesario

test('Un login exitoso', () => {
    const fintech = new Fintech();
    fintech.crearUsuario('usuario2', 'contraseña2');
    expect(fintech.login('usuario2', 'contraseña2')).toBe(true);
});
```

Login fallido

```
// loginFallido.test.js
const { Fintech } = require('./path-to-your-classes'); // Ajusta el path según sea necesario

test('Un login fallido', () => {
    const fintech = new Fintech();
    fintech.crearUsuario('usuario3', 'contraseña3');
    expect(fintech.login('usuario3', 'wrongpassword')).toBe(false);
});
```

Crear un usuario e ingresar dinero

```
// crearUsuarioEIngresarDinero.test.js
const { Fintech, Cuenta } = require('./path-to-your-classes'); // Ajusta el path según sea necesario

test('Crear un usuario e ingresar dinero', () => {
    const fintech = new Fintech();
    fintech.crearUsuario('usuario4', 'contraseña4');
    const cuenta = new Cuenta('1234', 'usuario4');
    fintech.agregarCuenta('usuario4', cuenta);
    cuenta.depositar(100, 'Ingreso inicial');
    expect(cuenta.consultarSaldo()).toBe(100);
});
```

Crear dos usuarios, ingresar dinero y transferir de una cuenta a la otra

```
// transferirDinero.test.js
const { Fintech, Cuenta } = require('./path-to-your-classes'); // Ajusta el path según sea necesario

test('Crear dos usuarios, ingresar dinero y transferir de una cuenta a la otra', () => {
    const fintech = new Fintech();
    fintech.crearUsuario('usuario5', 'contraseña5');
    fintech.crearUsuario('usuario6', 'contraseña6');
    const cuenta1 = new Cuenta('5678', 'usuario5');
    const cuenta2 = new Cuenta('91011', 'usuario6');
    fintech.agregarCuenta('usuario5', cuenta1);
    fintech.agregarCuenta('usuario6', cuenta2);
    cuenta1.depositar(200, 'Ingreso inicial');
    cuenta1.transferir(50, cuenta2, 'Transferencia');
    expect(cuenta1.consultarSaldo()).toBe(150);
    expect(cuenta2.consultarSaldo()).toBe(50);
});
```