**Documento de Diseño de Software**

**Aplicación Móvil de Lealtad LFDC**

## **1. Visión General**

Este documento especifica la estructura de software necesaria para implementar la aplicación móvil de lealtad LFDC, subdividida en dos niveles de diseño:

1. **Diseño Arquitectónico (Alto Nivel)** Describe los componentes globales, sus interacciones y consideraciones clave (desempeño, interfaces, seguridad, base de datos, manejo de errores).
2. **Diseño Detallado (Bajo Nivel)** Proporciona la especificación concreta de cada módulo/componente (formatos de entrada/salida, estructuras de datos, convenciones de nombres, especificaciones de almacenamiento, pseudocódigo).

Se omiten diagramas gráficos; en su lugar, se describen textualmente todas las relaciones y estructuras.

## **2. Diseño Arquitectónico**

### **2.1. Componentes Principales**

1. **Aplicación Móvil (Cliente)**
   * Framework: React Native ≥ 0.64 (compatible con Android 7.0+ e iOS 12+).
   * Módulos internos:  
     1. Pantallas de Autenticación (OAuth Facebook y registro manual).
     2. Vista de Saldo y Historial de Puntos/Recargas.
     3. Catálogo de Recompensas y Canje de Puntos.
     4. Formulario de Recarga con integración de pasarela de pago.
     5. Módulo de Notificaciones (suscripción a FCM/APNs).
2. **Backend RESTful / API**
   * Lenguaje/Entorno:  
     1. Opción primaria: Node.js ≥ 14 LTS (alternativa: Java ≥ 11).
   * Framework sugerido:  
     1. Express (Node.js) o Spring Boot (Java).
   * Módulos internos:  
     1. **Autenticación & Autorización**
        + Endpoints: /api/auth/register, /api/auth/login, /api/auth/logout, /api/auth/delete-account.
        + Soporta JWT y OAuth (Facebook).
     2. **Gestión de Usuario y Perfil**
        + Endpoints: /api/users/{id}, /api/users/{id}/update, /api/users/{id}/delete.
     3. **Gestión de Puntos**
        + Endpoints:  
          - Crear transacción: /api/transactions/create.
          - Consultar historial de transacciones: /api/transactions/{userId} (paginado).
          - Consultar puntos totales: /api/points/{userId}.
     4. **Canje de Recompensas**
        + Endpoints:  
          - Listar catálogo: /api/rewards.
          - Solicitar canje: /api/rewards/redeem.
          - Historial de canjes: /api/rewards/history/{userId}.
     5. **Recarga de Saldo**
        + Endpoints:  
          - Solicitar recarga (crea registro de recarga): /api/recharges/create.
          - Callback de pasarela de pago: /api/recharges/notify (Stripe/PayPal webhook).
          - Consultar historial de recargas: /api/recharges/{userId}.
     6. **Notificaciones Push**
        + Endpoint para envío manual/masivo: /api/notifications/send.
        + Suscripción de token: /api/notifications/subscribe.
        + Historial de notificaciones: /api/notifications/history/{userId} (para Admin).
     7. **Módulo de Gerente de Sucursal**
        + Endpoints:  
          - Login del gerente: /api/manager/login.
          - Listar solicitudes de canje pendientes en sucursal: /api/manager/branch/{branchId}/pending-redemptions.
          - Aprobar/rechazar canje: /api/manager/redeem/{redeemId}/approve y /api/manager/redeem/{redeemId}/reject.
          - Consultar reportes de uso:

/api/manager/branch/{branchId}/report/points-redeemed?period=day|week|month.

/api/manager/branch/{branchId}/report/recharges?period=day|week|month.

* + 1. **Módulo de Administrador del Sistema**
       - Endpoints:  
         * Login admin con TFA: /api/admin/login.
         * CRUD Políticas de Puntos: /api/admin/points-policy.
         * CRUD Recompensas: /api/admin/rewards.
         * Gestión de usuarios/roles: /api/admin/users, /api/admin/users/{id}/role.
         * Consultar historial global de notificaciones: /api/admin/notifications/history.
         * Reportes globales:

/api/admin/report/points-redeemed.

/api/admin/report/recharges.

1. **Base de Datos Relacional (PostgreSQL)**
   * Instancia escalable, con backups diarios y retención de 30 días.
   * Tablas principales:  
     1. users
     2. transactions
     3. points\_balance (o efecto derivado de sumatorias en transactions)
     4. rewards
     5. redemptions
     6. recharges
     7. branches (sucursales físicas)
     8. managers (credenciales de gerentes)
     9. admins (credenciales de administradores + TFA secreto)
     10. notifications\_log
     11. points\_policy
2. **Servicio de Notificaciones (Push)**
   * **Firebase Cloud Messaging (FCM)** para Android.
   * **Apple Push Notification Service (APNs)** para iOS.
   * Un microservicio o módulo dentro del Backend que envíe a FCM/APNs según token y plataforma, reutilizando librerías oficiales (firebase-admin para Node.js; nodemailer para notificaciones por correo/SMS).
3. **Pasarela de Pago (Stripe / PayPal)**
   * Ambiente sandbox para pruebas y modo live en producción.
   * Se configura webhook para recibir notificación de pago exitoso/fallido.

### **2.2. Relaciones entre Componentes**

* El **Cliente Móvil** consume exclusivamente el Backend vía HTTPS.
* El **Backend** interactúa con:  
  + **Base de Datos PostgreSQL** (con pool de conexiones).
  + **Servicio de Notificaciones** (lógica interna para disparar a FCM/APNs).
  + **Pasarela de Pago** (Stripe/PayPal) a través de SDKs y webhooks.
* El **Módulo de Gerente** y **Módulo de Admin** comparten la infra­estructura backend, pero con roles y scopes de permisos distintos (JWT con claims específicos).
* Todos los servicios corren detrás de un **Load Balancer** (p.ej., AWS ALB o NGINX) con certificados TLS 1.2+.

### **2.3. Consideraciones de Desempeño**

* **Pool de Conexiones a DB**: limitar máximos (p. ej., 100 conexiones) y usar caché en consultas frecuentes (e.g., saldo de puntos).
* **Caché**:  
  + Utilizar Redis (o Memcached) para la consulta de “Saldo de puntos” con TTL = 30 min.
  + Posible almacenamiento en Redis de tokens de sesión y TFA.
* **Rate Limiting**:  
  + Máximo 5 solicitudes/segundo por usuario en endpoints públicos (implementado con middleware como express-rate-limit).
* **Escalabilidad**:  
  + Backend stateless, escalable horizontalmente.
  + Base de datos replicada en lectura si el tráfico lo exige.

### **2.4. Interfaces**

1. **Interfaces de Software (APIs)**
   * Todos los endpoints exponen JSON.
   * Estándar de respuestas:

{

"success": true|false,

"data": { … },

"error": {

"code": "ERROR\_CODE",

"message": "Mensaje descriptivo"

}

}

1. **Interfaces de Hardware**
   * Teléfonos móviles Android 7.0+ e iOS 12+: cámaras (para perfil), GPS (solo si se decide usar geolocalización de sucursal), persistencia local (SQLite o AsyncStorage).
   * Comunicación HTTPS/TLS 1.2+ con servidor.
2. **Interfaces Humanas**
   * Pantallas de la App Mobile con UX intuitivo (botón “Atrás” consistente en todas las vistas).
   * Consolas web (Dashboard web) para gerente y admin, optimizadas para escritorio (≥ 1024 px de ancho), accesibilidad WCAG: contraste ≥ 4.5:1, labels accesibles.

### **2.5. Consideraciones de Seguridad**

1. **En Tránsito**
   * TLS 1.2+ forzado en todos los endpoints.
   * HSTS y cabeceras de seguridad (CSP, XSS Protection).
2. **En Reposo**
   * Contraseñas de usuarios y gerentes cifradas con bcrypt (work factor ≥ 12).
   * Tokens JWT firmados con clave secreta de al menos 256 bits, almacenada en vault/secret manager.
   * Datos sensibles (número de tarjeta) nunca almacenados completos: solo token obtenido de Stripe/PayPal.
3. **Autenticación y Autorización**
   * Clientes: JWT con expiración de sesión en 30 min de inactividad.
   * Gerentes/Admins: MFA obligatorio (SMS o App Authenticator).
   * Uso de middleware de autorización en cada endpoint para validar rol y scope (p.ej., verifyToken, checkRole('admin')).
4. **Protección LFPDPPP**
   * Consentimiento explícito en registro (pantalla con checkbox y enlace a política de privacidad).
   * Almacenamiento de bitácora de consentimiento (tabla user\_privacy\_consent).
   * Eliminación segura de datos si el usuario elimina cuenta (cascade en tablas, wiping de datos sensibles).

### **2.6. Diseño de Base de Datos (Resumen)**

#### **2.6.1. Tabla:**

#### **users**

* **Campos**:  
  + id (PK, UUID)
  + full\_name (VARCHAR(100), NO NULL)
  + email (VARCHAR(100), UNIQUE, NO NULL)
  + password\_hash (VARCHAR(255), NO NULL)
  + facebook\_id (VARCHAR(50), NULLABLE)
  + phone\_number (VARCHAR(20), NULLABLE)
  + profile\_photo\_url (VARCHAR(255), NULLABLE)
  + role (ENUM: ‘client’, ‘manager’, ‘admin’)
  + status (ENUM: ‘active’, ‘deleted’, ‘disabled’)
  + created\_at (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())
  + updated\_at (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())

#### **2.6.2. Tabla:**

#### **transactions**

* **Campos**:  
  + id (PK, UUID)
  + user\_id (FK → users.id, NO NULL)
  + amount\_spent (NUMERIC(10,2), NO NULL)
  + points\_earned (INT, NO NULL)
  + transaction\_date (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())
  + type (ENUM: ‘purchase’, ‘recharge’, ‘redemption’, ‘adjustment’)

#### **2.6.3. Tabla:**

#### **points\_balance**

* **Campos**:  
  + user\_id (PK, FK → users.id)
  + current\_points (INT, NO NULL)
  + points\_expires\_at (TIMESTAMP WITH TIME ZONE, NULLABLE)
  + last\_updated (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())
* *(Alternativamente, current\_points puede derivarse sumando points\_earned de transactions y restando canjes.)*

#### **2.6.4. Tabla:**

#### **rewards**

* **Campos**:  
  + id (PK, UUID)
  + name (VARCHAR(100), NO NULL)
  + description (TEXT, NULLABLE)
  + cost\_in\_points (INT, NO NULL)
  + image\_url (VARCHAR(255), NULLABLE)
  + stock\_available (INT, NO NULL)
  + created\_at, updated\_at (TIMESTAMPs)

#### **2.6.5. Tabla:**

#### **redemptions**

* **Campos**:  
  + id (PK, UUID)
  + user\_id (FK → users.id, NO NULL)
  + reward\_id (FK → rewards.id, NO NULL)
  + branch\_id (FK → branches.id, NULLABLE)
  + points\_deducted (INT, NO NULL)
  + redemption\_date (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())
  + status (ENUM: ‘pending’, ‘approved’, ‘rejected’)
  + reject\_reason (TEXT, NULLABLE)

#### **2.6.6. Tabla:**

#### **recharges**

* **Campos**:  
  + id (PK, UUID)
  + user\_id (FK → users.id, NO NULL)
  + amount (NUMERIC(10,2), NO NULL)
  + status (ENUM: ‘pending’, ‘completed’, ‘failed’)
  + payment\_provider (ENUM: ‘stripe’, ‘paypal’)
  + provider\_reference (VARCHAR(100), NULLABLE)
  + created\_at, updated\_at (TIMESTAMPs)

#### **2.6.7. Tabla:**

#### **branches**

* **Campos**:  
  + id (PK, UUID)
  + name (VARCHAR(100), NO NULL)
  + location (VARCHAR(255), NULLABLE)

#### **2.6.8. Tabla:**

#### **managers**

* **Campos**:  
  + id (PK, UUID)
  + user\_id (FK → users.id, NO NULL, con role = 'manager')
  + branch\_id (FK → branches.id, NO NULL)

#### **2.6.9. Tabla:**

#### **admins**

* **Campos**:  
  + id (PK, UUID)
  + user\_id (FK → users.id, NO NULL, con role = 'admin')
  + tfa\_secret (VARCHAR(100), NO NULL)
  + tfa\_enabled (BOOLEAN, DEFAULT TRUE)

#### **2.6.10. Tabla:**

#### **notifications\_log**

* **Campos**:  
  + id (PK, UUID)
  + recipient (VARCHAR(100), ‘ALL’, ‘SEGMENTO’ o user\_id)
  + title (VARCHAR(100), NO NULL)
  + body (TEXT, NO NULL)
  + data\_payload (JSONB, NULLABLE)
  + sent\_at (TIMESTAMP WITH TIME ZONE, DEFAULT NOW())
  + type (ENUM: ‘general’, ‘reward’, ‘expire\_reminder’, ‘flash’)

#### **2.6.11. Tabla:**

#### **points\_policy**

* **Campos**:  
  + id (PK, UUID)
  + conversion\_rate (NUMERIC(5,2), p. ej., 0.10 → 1 punto/$10)
  + expiration\_days (INT, valores permitidos 30|60|90)
  + created\_at, updated\_at (TIMESTAMPs)

### **2.7. Manejo de Errores y Atributos de Recuperación**

1. **Errores a Nivel de API**
   * Estructura de respuesta uniforme con campo error.
   * Códigos HTTP coherentes (400 para petición malformada; 401/403 para autenticación/permiso; 404 para no encontrado; 500 para errores internos).
   * Logueo en backend: nivel ERROR con stacktrace y request payload (sin datos sensibles).
   * Métricas de errores expuestos a sistema de monitoreo (p.ej., Prometheus + Grafana).
2. **Errores a Nivel de Base de Datos**
   * Transacciones atómicas en operaciones críticas (canje de puntos, recarga de saldo).
   * Rollback automático si falla alguna parte de la transacción.
   * Manejadores de colisión de clave (p.ej., si email duplicado, se lanza excepción capturada y se retorna 409 Conflict).
3. **Recuperación**
   * Backups diarios de la base de datos, retención de 30 días.
   * Pruebas semanales de restauración en ambiente de staging.
   * En caso de falla de pasarela de pago:  
     + Mostrar mensaje claro en app → “Error en el pago, intente nuevamente”.
     + Endpoint webhook valida estado del pago; si timeout, reintenta lógica de verificación en segundo plano.
   * Fallback de notificaciones: si falla FCM/APNs, almacenar en notifications\_log y reintentar en siguiente ventana (cron job cada hora).

## **3. Diseño Detallado**

### **3.1. Convenciones Generales**

* **Convenciones de Nombres**
  + Tablas: snake\_case en plural (users, transactions, rewards).
  + Columnas: snake\_case (full\_name, email).
  + Models/Entidades en código: PascalCase (User, Transaction).
  + Endpoints: kebab-case o snake\_case según lenguaje (/api/users/{id}/update).
  + Variables y funciones en Node.js: camelCase (getUserById, pointsPolicyService).
* **Formato de Estructuras de Datos (JSON)**
  + Ejemplo de usuario:

{

"id": "uuid-usuario",

"full\_name": "Luis González",

"email": "luis@example.com",

"profile\_photo\_url": "https://...",

"phone\_number": "5544332211"

}

* **Convenciones de Branches en Git**
  + feature/<descripcion>, bugfix/<descripcion>, hotfix/<descripcion>.
  + Commits con formato “semantic”:  
    - feat(auth): agregar login con Facebook OAuth
    - fix(points): corregir cálculo de puntos expirados

### **3.2. Especificación de Componentes del Backend**

#### **3.2.1. Módulo de Autenticación & Autorización**

* **Funcionalidad**
  1. Registro manual (email + password).
  2. Login con email/contraseña.
  3. Login con Facebook OAuth (usar Facebook SDK en mobile; backend recibe access\_token, verifica con Graph API, vincula o crea usuario LFDC).
  4. Emisión de JWT con claims:  
     + sub: user\_id
     + role: client/manager/admin
     + iat, exp (exp. en 30 min).
* **Endpoints**
  1. POST /api/auth/register  
     + Input (JSON):

{

"full\_name": "string",

"email": "string",

"password": "string"

}

* + - Output exitoso:

{

"success": true,

"data": {

"id": "uuid",

"full\_name": "string",

"email": "string",

"token": "jwt\_token"

}

}

* + - Errores:  
      * 400: Datos faltantes o formato inválido.
      * 409: Email ya registrado.
  1. POST /api/auth/login  
     + Input (JSON):

{

"email": "string",

"password": "string"

}

* + - Output exitoso similar a registro.
    - Errores:  
      * 400: Campos vacíos.
      * 401: Credenciales inválidas.
  1. POST /api/auth/facebook  
     + Input (JSON):

{

"facebook\_token": "string"

}

* + - Lógica:  
      1. Verificar token con Graph API de Facebook → obtener facebook\_id y email.
      2. Si users.facebook\_id = facebook\_id existe → emitir JWT.
      3. Si no existe pero email existe en users sin facebook\_id → actualizar con facebook\_id y emitir JWT.
      4. Si no existe email → crear nuevo usuario con facebook\_id.
    - Output exitoso:

{

"success": true,

"data": {

"id": "uuid",

"full\_name": "string",

"email": "string",

"token": "jwt\_token"

}

}

* 2. POST /api/auth/logout  
     + Lógica: invalidar token (opcional: blacklist en Redis).
     + Input: JWT en header Authorization: Bearer <token>.
     + Output: { "success": true }.
  3. DELETE /api/auth/delete-account  
     + Input: JWT.
     + Lógica:  
       1. Verificar identidad.
       2. Marcar users.status = 'deleted'.
       3. Cascade en transacciones/puntos (migrar a usuario fantasma opcional).
     + Output: { "success": true }.
* **Estructuras Internas (Pseudocódigo)**

// authService.js

async function registerUser(full\_name, email, password) {

if (!full\_name || !email || !password) throw new BadRequestError("Campos requeridos faltantes");

const existing = await UserRepo.findByEmail(email);

if (existing) throw new ConflictError("Email ya registrado");

const hash = await bcrypt.hash(password, 12);

const user = await UserRepo.create({ full\_name, email, password\_hash: hash, role: 'client', status: 'active' });

const token = jwt.sign({ sub: user.id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '30m' });

return { id: user.id, full\_name: user.full\_name, email: user.email, token };

}

async function loginUser(email, password) {

const user = await UserRepo.findByEmail(email);

if (!user || user.status !== 'active') throw new UnauthorizedError("Credenciales inválidas");

const valid = await bcrypt.compare(password, user.password\_hash);

if (!valid) throw new UnauthorizedError("Credenciales inválidas");

const token = jwt.sign({ sub: user.id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '30m' });

return { id: user.id, full\_name: user.full\_name, email: user.email, token };

}

async function loginWithFacebook(facebookToken) {

const fbData = await FacebookAPI.verifyToken(facebookToken);

if (!fbData || !fbData.email) throw new UnauthorizedError("Token Facebook inválido");

let user = await UserRepo.findByFacebookId(fbData.id);

if (!user) {

const byEmail = await UserRepo.findByEmail(fbData.email);

if (byEmail) {

user = await UserRepo.update(byEmail.id, { facebook\_id: fbData.id });

} else {

user = await UserRepo.create({

full\_name: fbData.name,

email: fbData.email,

facebook\_id: fbData.id,

password\_hash: null,

role: 'client',

status: 'active'

});

}

}

const token = jwt.sign({ sub: user.id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '30m' });

return { id: user.id, full\_name: user.full\_name, email: user.email, token };

}

#### **3.2.2. Módulo de Gestión de Perfil de Cliente**

* **Funcionalidad**
  + Obtener datos: GET /api/users/{id}.
  + Actualizar datos: PUT /api/users/{id}/update.
  + Cambiar foto de perfil (endpoint que acepta multipart/form-data).
  + Cerrar sesión: invocar POST /api/auth/logout.
  + Eliminar cuenta: DELETE /api/auth/delete-account.
* **Formato de Input/Output**
  + **GET /api/users/{id}**
    - Input: JWT en header.
    - Output:

{

"success": true,

"data": {

"id": "uuid",

"full\_name": "string",

"email": "string",

"profile\_photo\_url": "string",

"phone\_number": "string|null"

}

}

* + **PUT /api/users/{id}/update**
    - Input (JSON):

{

"full\_name": "string",

"phone\_number": "string|"null",

"profile\_photo\_url": "string|"null"

}

* + - Output: { "success": true, "data": { …usuario actualizado… } }.
* **Pseudocódigo de Actualización**

async function updateUserProfile(userId, { full\_name, phone\_number, profile\_photo\_url }) {

const existing = await UserRepo.findById(userId);

if (!existing || existing.status !== 'active') throw new NotFoundError("Usuario no encontrado");

const updated = await UserRepo.update(userId, { full\_name, phone\_number, profile\_photo\_url, updated\_at: new Date() });

return updated;

}

#### **3.2.3. Módulo de Acumulación y Consulta de Puntos**

* **Flujo de Acumulación**
  1. Cada vez que el cliente realiza una compra en el sistema de punto de venta (POS), se envía al backend un request:  
       
      POST /api/transactions/create  
     + Input (JSON):

{

"user\_id": "uuid",

"amount\_spent": 150.00,

"transaction\_type": "purchase"

}

* + - Backend calcula puntos según política (e.g., points = floor(amount\_spent / 10)).
    - Inserta en la tabla transactions.
    - Actualiza o inserta registro en points\_balance:  
      * current\_points += points
      * points\_expires\_at = NOW() + INTERVAL 'policy.expiration\_days days'
    - Output:

{

"success": true,

"data": {

"transaction\_id": "uuid-transaccion",

"points\_earned": 15,

"current\_points": 200

}

}

* **Consultar Historial y Saldo**
  + **GET /api/transactions/{userId}**
    - Parámetros de query: page, limit.
    - Output:

{

"success": true,

"data": {

"transactions": [

{

"id": "uuid",

"amount\_spent": 150.00,

"points\_earned": 15,

"transaction\_date": "2025-06-02T14:30:00Z"

},

…

],

"pagination": { "page": 1, "limit": 20, "total": 45 }

}

}

* + **GET /api/points/{userId}**
    - Output:

{

"success": true,

"data": {

"current\_points": 200,

"points\_expires\_at": "2025-08-01T00:00:00Z"

}

}

* **Pseudocódigo (Creación de Transacción)**

async function createTransaction({ user\_id, amount\_spent, transaction\_type }) {

const policy = await PointsPolicyRepo.getActive();

const points = Math.floor(amount\_spent / (1 / policy.conversion\_rate));

const tx = await TransactionRepo.create({

user\_id,

amount\_spent,

points\_earned: points,

transaction\_date: new Date(),

type: 'purchase'

});

const balance = await PointsBalanceRepo.findByUserId(user\_id)

?? await PointsBalanceRepo.create({ user\_id, current\_points: 0, points\_expires\_at: null });

const newPoints = balance.current\_points + points;

const newExpiry = new Date(Date.now() + policy.expiration\_days \* 24 \* 60 \* 60 \* 1000);

await PointsBalanceRepo.update(user\_id, { current\_points: newPoints, points\_expires\_at: newExpiry, last\_updated: new Date() });

return {

transaction\_id: tx.id,

points\_earned: points,

current\_points: newPoints

};

}

#### **3.2.4. Módulo de Canje de Puntos**

* **Flujo de Canje**
  1. Cliente consulta catálogo: GET /api/rewards  
     + Output:

{

"success": true,

"data": [

{

"id": "uuid",

"name": "Café Gratis",

"description": "Taza de café gratis",

"cost\_in\_points": 100,

"image\_url": "https://...",

"stock\_available": 50

},

…

]

}

* 2. Al seleccionar la recompensa, se invoca: POST /api/rewards/redeem  
     + Input (JSON):

{

"user\_id": "uuid",

"reward\_id": "uuid",

"branch\_id": "uuid-sucursal"

}

* + - Validaciones:  
      * Verificar points\_balance.current\_points >= cost\_in\_points.
      * Verificar rewards.stock\_available > 0.
    - Operación atómica:  
      * Insertar en redemptions con status = 'pending'.
      * Reducir points\_balance.current\_points en cost\_in\_points.
      * Reducir rewards.stock\_available en 1.
      * Notificar al sistema de gerentes (p. ej., enviar mensaje a cola interna o WebSocket) para que sucursal correspondiente procese el canje.
    - Output:

{

"success": true,

"data": {

"redemption\_id": "uuid",

"new\_points\_balance": 1000

}

}

* 2. Gerente aprueba/rechaza:  
     + **PUT /api/manager/redeem/{redeemId}/approve**
       - Input: {} (sólo JWT con rol manager).
       - Lógica: actualizar redemptions.status = 'approved' y updated\_at.
       - Notificar al cliente vía push/SMS/correo (p. ej., payload FCM: { title: "Canje aprobado", body: "Tu canje de Café Gratis ha sido aprobado" }).
       - Output: { "success": true }.
     + **PUT /api/manager/redeem/{redeemId}/reject**
       - Input (JSON):

{

"reason": "Stock insuficiente"

}

* + - * Lógica:  
        1. Actualizar redemptions.status = 'rejected', reject\_reason = reason.
        2. Reembolsar puntos: incrementar en points\_balance.current\_points la cantidad points\_deducted.
        3. Incrementar rewards.stock\_available en 1.
      * Output: { "success": true }.
  1. Cliente consulta historial de canjes: GET /api/rewards/history/{userId}  
     + Paginado.
     + Output:

{

"success": true,

"data": {

"redemptions": [

{

"id": "uuid",

"reward\_name": "Café Gratis",

"points\_deducted": 100,

"status": "approved",

"redemption\_date": "2025-05-30T10:00:00Z"

},

…

],

"pagination": { … }

}

}

#### **3.2.5. Módulo de Recargas de Saldo**

* **Flujo de Recarga**
  1. Cliente abre pantalla “Recargar Saldo” y llena:  
     + amount (≥ 50).
     + Datos de tarjeta (manejado por Stripe/PayPal SDK en mobile).
  2. App envía: POST /api/recharges/create  
     + Input (JSON):

{

"user\_id": "uuid",

"amount": 100.00,

"payment\_method": "stripe"|"paypal"

}

* + - Lógica:  
      1. Crear registro en recharges con status = 'pending'.
      2. Generar un pago con Stripe/PayPal: obtener provider\_reference (p.ej., checkout\_session\_id).
      3. Retornar al cliente datos necesarios para completar el pago (p.ej., checkout\_url).
    - Output:

{

"success": true,

"data": {

"recharge\_id": "uuid",

"checkout\_url": "https://checkout.stripe.com/…"

}

}

* 2. Cliente completa el pago en sandbox o Producción. Stripe/PayPal invoca webhook en el backend:  
     + Endpoint: POST /api/recharges/notify
     + Payload (Stripe ejemplo):

{

"id": "evt\_1ABCdeFGHIjklMN",

"type": "checkout.session.completed",

"data": {

"object": {

"id": "cs\_test\_123",

"amount\_total": 10000,

"metadata": {

"recharge\_id": "uuid"

}

}

}

}

* + - Lógica del webhook:  
      1. Verificar firma (Stripe Signature Header).
      2. Obtener recharge\_id desde metadata.
      3. Actualizar recharges.status = 'completed', provider\_reference = cs\_test\_123.
      4. Actualizar saldo de recarga del usuario (tabla transactions con tipo recharge o directamente en points\_balance si se maneja un saldo en pesos separado).
      5. Enviar notificación de confirmación al cliente (push/email).
  1. Cliente consulta historial de recargas: GET /api/recharges/{userId}  
     + Paginado, muestra amount, status, created\_at.
* **Pseudocódigo (Webhook de Stripe)**

async function handleStripeWebhook(req, res) {

const sig = req.headers['stripe-signature'];

let event;

try {

event = stripe.webhooks.constructEvent(req.rawBody, sig, process.env.STRIPE\_WEBHOOK\_SECRET);

} catch (err) {

return res.status(400).send(`Webhook Error: ${err.message}`);

}

if (event.type === 'checkout.session.completed') {

const session = event.data.object;

const rechargeId = session.metadata.recharge\_id;

await RechargeRepo.update(rechargeId, {

status: 'completed',

provider\_reference: session.id,

updated\_at: new Date()

});

// Registrar en transactions

await TransactionRepo.create({

user\_id: session.metadata.user\_id,

amount\_spent: session.amount\_total / 100,

points\_earned: 0,

transaction\_date: new Date(),

type: 'recharge'

});

// Notificar al usuario

NotificationService.send({

recipient: session.metadata.user\_id,

title: "Recarga exitosa",

body: `Se han agregado $${ session.amount\_total / 100 } a tu saldo.`,

data: { type: 'recharge\_success' }

});

}

res.json({ received: true });

}

#### **3.2.6. Módulo de Notificaciones**

* **Tipos**
  + **Promociones generales**: e.g., “2×1 en Frappes”.
  + **Recompensas disponibles**: publicación de nuevas recompensas en catálogo.
  + **Recordatorio de puntos próximos a caducar**: enviar 7 días antes de points\_expires\_at.
  + **Alertas relámpago**: descuento válido solo 2 horas (generar notificaciones con TTL interno o cron job que remueva).
* **Endpoint para envío**
  + POST /api/notifications/send  
    - Input (JSON):

{

"recipient": "user\_id" | "ALL" | "SEGMENTO",

"title": "string",

"body": "string",

"data": { "additional": "payload\_opcional" }

}

* + - Lógica:  
      1. Si recipient = 'ALL', obtener lista de todos los usuarios activos con tokens válidos.
      2. Si SEGMENTO, aplicar filtro (p.ej., usuarios con puntos ≥ 500).
      3. Para cada token de dispositivo, invocar FCM/APNs.
      4. Insertar registro en notifications\_log.
    - Output: { "success": true, "data": { "sent\_count": Número } }.
* **Subscripción de Tokens**
  + POST /api/notifications/subscribe  
    - Input:

{

"user\_id": "uuid",

"platform": "android"|"ios",

"token": "string\_device\_token"

}

* + - Lógica:  
      * Guardar en tabla device\_tokens con user\_id, platform, token, created\_at.
* **Proceso Automático de Vencimiento de Puntos**
  + Tarea programada (cron job diario a las 00:00):  
    - Consultar points\_balance donde points\_expires\_at = fecha\_hoy + 7 días.
    - Para cada user\_id, invocar /api/notifications/send con recordatorio de expiración.

#### **3.2.7. Módulo de Gerente de Sucursal**

* **Autenticación**
  1. POST /api/manager/login  
     + Similar a /api/auth/login pero solo para role = 'manager'.
* **Validar Canjes**
  1. **GET /api/manager/branch/{branchId}/pending-redemptions**
     + Output:

{

"success": true,

"data": {

"redemptions": [

{

"id": "uuid",

"user\_name": "string",

"reward\_name": "string",

"points\_deducted": 100,

"redemption\_date": "2025-06-01T09:00:00Z"

},

…

]

}

}

* 2. **PUT /api/manager/redeem/{redeemId}/approve**
     + Actualiza redemptions.status = 'approved'.
     + Output: { "success": true }.
  3. **PUT /api/manager/redeem/{redeemId}/reject**
     + Input:

{ "reason": "string" }

* + - Lógica: ver sección 3.2.4.
* **Reportes de Uso de Sucursal**
  1. **GET /api/manager/branch/{branchId}/report/points-redeemed?period=day|week|month**
     + Cálculo:

SELECT

DATE\_TRUNC('day', redemption\_date) AS period,

SUM(points\_deducted) AS total\_points

FROM redemptions

WHERE branch\_id = :branchId

AND status = 'approved'

AND redemption\_date >= NOW() - INTERVAL '1 {period}'

GROUP BY 1;

* + - Output:

{

"success": true,

"data": [

{ "period": "2025-06-01", "total\_points": 500 },

…

]

}

* 2. **GET /api/manager/branch/{branchId}/report/recharges?period=day|week|month**
     + Cálculo:

SELECT

DATE\_TRUNC('day', created\_at) AS period,

COUNT(\*) AS total\_recharges,

SUM(amount) AS total\_amount

FROM recharges

WHERE branch\_id = :branchId

AND status = 'completed'

AND created\_at >= NOW() - INTERVAL '1 {period}'

GROUP BY 1;

* + - Output: similar al anterior.

#### **3.2.8. Módulo de Administrador del Sistema**

* **Autenticación Admin**
  + POST /api/admin/login  
    - Input: email + password + tfa\_code.
    - Lógica:  
      1. Verificar credenciales.
      2. Verificar TFA con tfa\_secret (usando algoritmo TOTP).
      3. Emitir JWT con role = 'admin'.
* **Gestión de Políticas de Puntos**
  + **GET /api/admin/points-policy**
    - Output: parámetros actuales (conversion\_rate, expiration\_days).
  + **POST /api/admin/points-policy**
    - Input:

{

"conversion\_rate": 0.10,

"expiration\_days": 60

}

* + - Lógica: actualizar registro activo en points\_policy.
* **Administrar Recompensas**
  + **GET /api/admin/rewards**
    - Listar todas (con filtros opcionales: nombre, stock < x).
  + **POST /api/admin/rewards**
    - Input:

{

"name": "string",

"description": "string",

"cost\_in\_points": 100,

"image\_url": "string",

"stock\_available": 50

}

* + - Crea un nuevo producto en rewards.
  + **PUT /api/admin/rewards/{rewardId}**
    - Actualiza campos.
  + **DELETE /api/admin/rewards/{rewardId}**
    - Elimina/retiro lógico (status = 'inactive' opcional).
* **Administrar Usuarios y Roles**
  + **GET /api/admin/users**
    - Parámetros de query: role, status, page, limit.
    - Output con lista de usuarios y sus roles.
  + **PUT /api/admin/users/{userId}/role**
    - Input:

{

"role": "client"|"manager"|"admin",

"status": "active"|"disabled"

}

* + - Actualiza rol y estado.
  1. **DELETE /api/admin/users/{userId}**
     + Marcar status = 'deleted'.
* **Monitoreo de Notificaciones**
  1. **GET /api/admin/notifications/history**
     + Parámetros: fromDate, toDate, recipient.
     + Output: registros de notifications\_log.
* **Reportes Globales**
  1. **GET /api/admin/report/points-redeemed**
     + Sumar puntos redimidos globalmente:

SELECT DATE\_TRUNC('day', redemption\_date) AS day, SUM(points\_deducted) AS total\_points

FROM redemptions

WHERE status = 'approved'

AND redemption\_date >= NOW() - INTERVAL '30 days'

GROUP BY 1;

* 2. **GET /api/admin/report/recharges**
     + Sumar recargas globales:

SELECT DATE\_TRUNC('day', created\_at) AS day, COUNT(\*) AS total\_recharges, SUM(amount) AS total\_amount

FROM recharges

WHERE status = 'completed'

AND created\_at >= NOW() - INTERVAL '30 days'

GROUP BY 1;

### **3.3. Diseño Detallado de la Base de Datos**

A continuación se detallan los campos, tipos y propósito de cada elemento de datos en las tablas principales.

#### **3.3.1. Tabla**

#### **users**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador único del usuario |
| full\_name | VARCHAR(100) | NO | Nombre completo |
| email | VARCHAR(100) | NO, UNIQUE | Correo electrónico único |
| password\_hash | VARCHAR(255) | NO | Hash bcrypt de la contraseña |
| facebook\_id | VARCHAR(50) | SÍ | ID de Facebook para OAuth |
| phone\_number | VARCHAR(20) | SÍ | Número de teléfono (opcional) |
| profile\_photo\_url | VARCHAR(255) | SÍ | URL de la foto de perfil |
| role | ENUM | NO | ‘client’, ‘manager’ o ‘admin’ |
| status | ENUM | NO | ‘active’, ‘deleted’, ‘disabled’ |
| created\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de última modificación |

#### **3.3.2. Tabla**

#### **transactions**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la transacción |
| user\_id | UUID | NO | FK → users.id |
| amount\_spent | NUMERIC(10,2) | NO | Monto gastado en la transacción (en moneda local) |
| points\_earned | INT | NO | Puntos ganados (puede ser cero en recarga, ajuste o redención) |
| transaction\_date | TIMESTAMP WITH TIME ZONE | NO | Fecha y hora de la transacción |
| type | ENUM | NO | ‘purchase’, ‘recharge’, ‘redemption’, ‘adjustment’ |

#### **3.3.3. Tabla**

#### **points\_balance**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| user\_id | UUID | NO (PK) | FK → users.id |
| current\_points | INT | NO | Puntos disponibles actualmente |
| points\_expires\_at | TIMESTAMP WITH TIME ZONE | SÍ | Fecha de expiración de los puntos (basado en política activa) |
| last\_updated | TIMESTAMP WITH TIME ZONE | NO | Fecha de última actualización |

#### **3.3.4. Tabla**

#### **rewards**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la recompensa |
| name | VARCHAR(100) | NO | Nombre de la recompensa |
| description | TEXT | SÍ | Descripción detallada |
| cost\_in\_points | INT | NO | Costo en puntos necesario para canjear |
| image\_url | VARCHAR(255) | SÍ | URL de la imagen representativa |
| stock\_available | INT | NO | Cantidad en inventario |
| created\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de última modificación |

#### **3.3.5. Tabla**

#### **redemptions**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la solicitud de canje |
| user\_id | UUID | NO | FK → users.id |
| reward\_id | UUID | NO | FK → rewards.id |
| branch\_id | UUID | SÍ | FK → branches.id (sucursal donde se canjea) |
| points\_deducted | INT | NO | Puntos deducidos en el canje |
| redemption\_date | TIMESTAMP WITH TIME ZONE | NO | Fecha de la petición de canje |
| status | ENUM | NO | ‘pending’, ‘approved’, ‘rejected’ |
| reject\_reason | TEXT | SÍ | Motivo de rechazo (si aplica) |

#### **3.3.6. Tabla**

#### **recharges**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la recarga |
| user\_id | UUID | NO | FK → users.id |
| amount | NUMERIC(10,2) | NO | Monto recargado |
| status | ENUM | NO | ‘pending’, ‘completed’, ‘failed’ |
| payment\_provider | ENUM | NO | ‘stripe’, ‘paypal’ |
| provider\_reference | VARCHAR(100) | SÍ | ID o referencia que devuelve el proveedor |
| created\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de última actualización |

#### **3.3.7. Tabla**

#### **branches**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la sucursal |
| name | VARCHAR(100) | NO | Nombre de la sucursal |
| location | VARCHAR(255) | SÍ | Dirección o descripción |

#### **3.3.8. Tabla**

#### **managers**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador interno |
| user\_id | UUID | NO | FK → users.id (debe tener role = 'manager') |
| branch\_id | UUID | NO | FK → branches.id |

#### **3.3.9. Tabla**

#### **admins**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador interno |
| user\_id | UUID | NO | FK → users.id (debe tener role = 'admin') |
| tfa\_secret | VARCHAR(100) | NO | Secreto TOTP para autenticación de dos factores |
| tfa\_enabled | BOOLEAN | NO | Indica si TFA está habilitado (true/false) |

#### **3.3.10. Tabla**

#### **notifications\_log**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador del registro de notificación |
| recipient | VARCHAR(100) | NO | user\_id, ‘ALL’ o ‘SEGMENTO’ |
| title | VARCHAR(100) | NO | Título del mensaje |
| body | TEXT | NO | Cuerpo del mensaje |
| data\_payload | JSONB | SÍ | Carga útil adicional (enlaces, tipos, etc.) |
| sent\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha y hora de envío |
| type | ENUM | NO | ‘general’, ‘reward’, ‘expire\_reminder’, ‘flash’ |

#### **3.3.11. Tabla**

#### **points\_policy**

| **Campo** | **Tipo** | **Nullable** | **Descripción** |
| --- | --- | --- | --- |
| id | UUID | NO | Identificador de la política activa |
| conversion\_rate | NUMERIC(5,2) | NO | Ejemplo: 0.10 → 1 punto / $10 |
| expiration\_days | INT | NO | Número de días para caducidad (30, 60, 90) |
| created\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE | NO | Fecha de última actualización |

### **3.4. Diseño Detallado de Datos e Interfaces**

#### **3.4.1. Formatos de Entrada / Salida de Datos**

* Todas las peticiones y respuestas usan **JSON UTF-8**.
* El cliente envía JWT en header:

Authorization: Bearer <token>

* Fechas en formato ISO 8601: "YYYY-MM-DDTHH:mm:ssZ".
* Errores devueltos con la siguiente estructura uniforme:

{

"success": false,

"error": {

"code": "ERROR\_CODE",

"message": "Mensaje descriptivo"

}

}

#### **3.4.2. Especificaciones de Necesidades de Almacenamiento**

* **Imagen de Perfil**:  
  + Almacenadas en servicio de Cloud Storage (p.ej., AWS S3), sólo se guarda URL en DB (profile\_photo\_url).
* **Imágenes de Recompensa**:  
  + Similar a perfil, con prefijo de bucket rewards/.
* **Tokens de Dispositivo (FCM/APNs)**:  
  + Tabla opcional device\_tokens con campos:  
    - id (UUID)
    - user\_id (UUID)
    - platform (android/ios)
    - token (VARCHAR(255))
    - created\_at (TIMESTAMP)
* **Archivos de Log**:  
  + Log de transacciones financieras, notificaciones, errores, en sistema de archivos o servicio de log (p.ej., ELK stack).

#### **3.4.3. Definición de Estructuras de Datos Requeridas**

1. **Objeto Usuario (User)**

type User = {

id: string; // UUID

full\_name: string; // Nombre completo

email: string; // Correo único

profile\_photo\_url?: string; // URL de foto (opcional)

phone\_number?: string; // Telefono (opcional)

role: 'client' | 'manager' | 'admin';

status: 'active' | 'deleted' | 'disabled';

created\_at: string; // ISO 8601

updated\_at: string; // ISO 8601

};

2. **Objeto Transacción (Transaction)**

type Transaction = {

id: string; // UUID

user\_id: string; // FK → User

amount\_spent: number; // Monto en moneda local

points\_earned: number; // Puntos obtenidos

transaction\_date: string; // ISO 8601

type: 'purchase' | 'recharge' | 'redemption' | 'adjustment';

};

2. **Objeto Saldo de Puntos (PointsBalance)**

type PointsBalance = {

user\_id: string; // FK → User

current\_points: number;

points\_expires\_at?: string; // ISO 8601 (opcional si nunca expiran)

last\_updated: string; // ISO 8601

};

2. **Objeto Recompensa (Reward)**

type Reward = {

id: string; // UUID

name: string;

description?: string;

cost\_in\_points: number;

image\_url?: string;

stock\_available: number;

created\_at: string; // ISO 8601

updated\_at: string; // ISO 8601

};

2. **Objeto Canje (Redemption)**

type Redemption = {

id: string; // UUID

user\_id: string; // FK → User

reward\_id: string; // FK → Reward

branch\_id?: string; // FK → Branch (opcional hasta aprobación)

points\_deducted: number;

redemption\_date: string; // ISO 8601

status: 'pending' | 'approved' | 'rejected';

reject\_reason?: string; // Sólo si status='rejected'

};

2. **Objeto Recarga (Recharge)**

type Recharge = {

id: string; // UUID

user\_id: string; // FK → User

amount: number; // Monto en moneda local

status: 'pending' | 'completed' | 'failed';

payment\_provider: 'stripe' | 'paypal';

provider\_reference?: string; // ID del proveedor de pago

created\_at: string; // ISO 8601

updated\_at: string; // ISO 8601

};

2. **Objeto Punto de Venta (Branch)**

type Branch = {

id: string; // UUID

name: string;

location?: string;

};

2. **Objeto Manager/Admin (Credentials)**

type Manager = {

id: string; // UUID

user\_id: string; // FK → User

branch\_id: string; // FK → Branch

};

type Admin = {

id: string; // UUID

user\_id: string; // FK → User

tfa\_secret: string;

tfa\_enabled: boolean;

};

2. **Objeto Notificación (NotificationLog)**

type NotificationLog = {

id: string; // UUID

recipient: string; // 'ALL', 'SEGMENTO' o user\_id

title: string;

body: string;

data\_payload?: Record<string, any>;

sent\_at: string; // ISO 8601

type: 'general' | 'reward' | 'expire\_reminder' | 'flash';

};

2. **Objeto Política de Puntos (PointsPolicy)**

type PointsPolicy = {

id: string; // UUID

conversion\_rate: number; // e.g., 0.10 → 1 punto / $10

expiration\_days: 30 | 60 | 90;

created\_at: string; // ISO 8601

updated\_at: string; // ISO 8601

};

### **3.5. Manejo de Errores y Recuperación (Detalles)**

1. **Errores de Validación (400)**
   * Campos faltantes, formato inválido, monto de recarga < 50, etc.
   * Ejemplo de respuesta:

{

"success": false,

"error": {

"code": "VALIDATION\_ERROR",

"message": "El campo 'amount' debe ser al menos 50"

}

}

2. **Errores de Autenticación (401/403)**
   * Token inválido o expirado.
   * Rol insuficiente:  
     + Ejemplo: manager accediendo a endpoint de admin.
   * Ejemplo de respuesta:

{

"success": false,

"error": {

"code": "UNAUTHORIZED",

"message": "Acceso no autorizado"

}

}

2. **Errores de Recursos No Encontrados (404)**
   * Usuario, transacción, recompensa no existe.
   * Ejemplo de respuesta:

{

"success": false,

"error": {

"code": "NOT\_FOUND",

"message": "Recompensa no encontrada"

}

}

2. **Errores de Lógica de Negocio (409)**
   * Puntos insuficientes al canjear.
   * Stock agotado de recompensa.
   * Ejemplo de respuesta:

{

"success": false,

"error": {

"code": "INSUFFICIENT\_POINTS",

"message": "No tienes suficientes puntos para canjear esta recompensa"

}

}

2. **Errores Internos (500)**
   * Excepción no manejada, fallo de conexión a DB.
   * Respuesta genérica:

{

"success": false,

"error": {

"code": "INTERNAL\_SERVER\_ERROR",

"message": "Error interno, por favor intenta más tarde"

}

}

2. **Recuperación / Mecanismos de Reintento**
   * Para webhooks de pago: reintento un máximo de 3 veces si falla la validación de firma antes de alertar a equipo de soporte.
   * Para envíos de notificaciones push:  
     + Si FCM/APNs retorna error temporal, reintentar 3 veces con backoff exponencial.
   * Tareas programadas (cron jobs):  
     + Enviar recordatorios de expiración de puntos, reintentar en caso de error y loguear al finalizar.

### **3.6. Módulo de Usabilidad y Accesibilidad (Cliente Móvil)**

1. **Tipografía y Contraste**
   * Texto principal: mínimo 16 px.
   * Contraste ≥ 4.5:1 en botones y textos.
2. **Flujos de Navegación**
   * Botón único “Atrás” en esquina superior izquierda para volver al menú principal.
   * Pantalla de login: accesible, con labels vinculados y placeholders descriptivos.
3. **Accesibilidad**
   * Todos los elementos interactivos con accessibilityLabel.
   * Compatibilidad con lectores de pantalla (VoiceOver, TalkBack).
   * Uso de TouchableOpacity con accesibilidad habilitada.
4. **Mensajes de Error / Feedback**
   * Validaciones en formulario (registro, recarga) muestran mensaje debajo del campo invalidado en color rojo y con icono de advertencia.

### **3.7. Módulo de Mantenibilidad y Documentación**

1. **Estructura de Carpetas Backend (Node.js)**

lfcd-backend/

├── src/

│ ├── controllers/

│ │ ├── authController.js

│ │ ├── userController.js

│ │ ├── pointsController.js

│ │ ├── rewardsController.js

│ │ ├── rechargesController.js

│ │ ├── notificationsController.js

│ │ ├── managerController.js

│ │ └── adminController.js

│ ├── services/

│ │ ├── authService.js

│ │ ├── userService.js

│ │ ├── pointsService.js

│ │ ├── rewardsService.js

│ │ ├── rechargeService.js

│ │ ├── notificationService.js

│ │ ├── managerService.js

│ │ └── adminService.js

│ ├── repositories/

│ │ ├── userRepo.js

│ │ ├── transactionRepo.js

│ │ ├── pointsBalanceRepo.js

│ │ ├── rewardRepo.js

│ │ ├── redemptionRepo.js

│ │ ├── rechargeRepo.js

│ │ ├── branchRepo.js

│ │ ├── managerRepo.js

│ │ ├── adminRepo.js

│ │ └── notificationLogRepo.js

│ ├── middlewares/

│ │ ├── authMiddleware.js

│ │ ├── rateLimitMiddleware.js

│ │ └── errorHandler.js

│ ├── models/ // Definición de esquemas ORM (p.ej., Sequelize o TypeORM)

│ ├── utils/

│ │ ├── jwtUtil.js

│ │ ├── bcryptUtil.js

│ │ └── validationUtil.js

│ ├── routes.js // Definición de rutas y mapeo a controladores

│ └── index.js // Punto de entrada, configuración de servidor, DB, middlewares

├── config/

│ ├── default.json // Configuración base (DB, keys, rate limit)

│ └── production.json

├── migrations/ // Archivos de migración de la base de datos

├── seeders/ // Datos iniciales (p.ej., política de puntos predeterminada)

├── tests/ // Pruebas unitarias e integración (Jest/Mocha)

└── README.md // Instrucciones para levantar localmente

2. **Consola de Gerente/Admin (Web)**
   * Estructura similar a la App Backend, pero con vistas en React (o Framework de elección).
   * Carpetas:

lfcd-dashboard/

├── public/

├── src/

│ ├── components/

│ ├── pages/

│ │ ├── loginManager.jsx

│ │ ├── pendingRedemptions.jsx

│ │ ├── reportPoints.jsx

│ │ ├── loginAdmin.jsx

│ │ ├── managePointsPolicy.jsx

│ │ ├── manageRewards.jsx

│ │ └── manageUsers.jsx

│ ├── services/ // Llama a endpoints del backend

│ ├── utils/

│ └── index.jsx

├── package.json

└── README.md

2. **Documentación Interna**
   * Cada función crítica con comentario JSDoc (> 5 líneas) explicando lógica y parámetros.
   * **README.md** explica:  
     + Requisitos previos (Node.js ≥ 14, PostgreSQL).
     + Variables de entorno obligatorias:  
       1. DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME, JWT\_SECRET, STRIPE\_SECRET\_KEY, STRIPE\_WEBHOOK\_SECRET, FCM\_SERVER\_KEY.
     + Pasos para levantar localmente:  
       1. git clone … && cd lfcd-backend
       2. cp config/default.json config/local.json (modificar credenciales).
       3. npm install
       4. npm run migrate && npm run seed
       5. npm run dev (arranca servidor en :3000).
   * **CHANGELOG.md** (generado automáticamente con convencional commits).
3. **Pruebas Automatizadas**
   * **Unitarias** (Jest/Mocha):  
     + Cobertura mínima del 80% en servicios críticos (authService, pointsService, rechargeService).
   * **Integración**:  
     + Test de flujos completos (registro→login→acumular puntos→canjear→aprobar).
     + Uso de base de datos en memoria (SQLite) o contenedor Docker para PostgreSQL.

### **3.8. Trazabilidad Requisito–Código**

* Cada requisito funcional se identifica con ID RF-XX, ligado a ticket en Trello/Jira.
* Ejemplo:  
  + RF-01: “Autenticación vía Facebook OAuth” → Ticket RF-01, commits asociados: [RF-01] Implementar login Facebook
  + RF-02: “Mostrar saldo de puntos al iniciar sesión” → Ticket RF-02
* Cada controlador verifica en comentario JSDoc el RF que cubre:

/\*\*

\* RF-02: Obtener saldo de puntos al login

\* GET /api/points/:userId

\*/

async function getPoints(req, res) { … }

## **4. Resumen de Consideraciones No Funcionales**

1. **Seguridad y Privacidad**
   * HTTPS obligatorio, cifrado bcrypt, JWT, MFA para roles sensibles.
   * Cumplimiento LFPDPPP con consentimiento explícito y políticas de privacidad.
2. **Rendimiento y Escalabilidad**
   * Caché Redis para consultas frecuentes (saldo de puntos).
   * Backend sin estado, escalable con load balancer.
   * Rate limiting: 5 req/s.
3. **Compatibilidad y Portabilidad**
   * Soporte Android 7.0+, iOS 12.
   * Adaptación Responsive en dashboard web.
4. **Disponibilidad y Confiabilidad**
   * Notificaciones 99.5% uptime.
   * Backups diarios, restauración en < 4 h.
   * Fallback de pago y reintentos en notificaciones.
5. **Usabilidad y Accesibilidad**
   * UI intuitiva: flujos consistentes de navegación.
   * Accesibilidad: etiquetas para lectores de pantalla, contraste 4.5:1.
6. **Mantenibilidad y Documentación**
   * Código documentado con JSDoc, README claro, tests automatizados, control de versiones con convención “semantic”.
7. **Trazabilidad**
   * Ticket por requisito, commits con referencia a ticket.
   * Changelog automático en cada release.