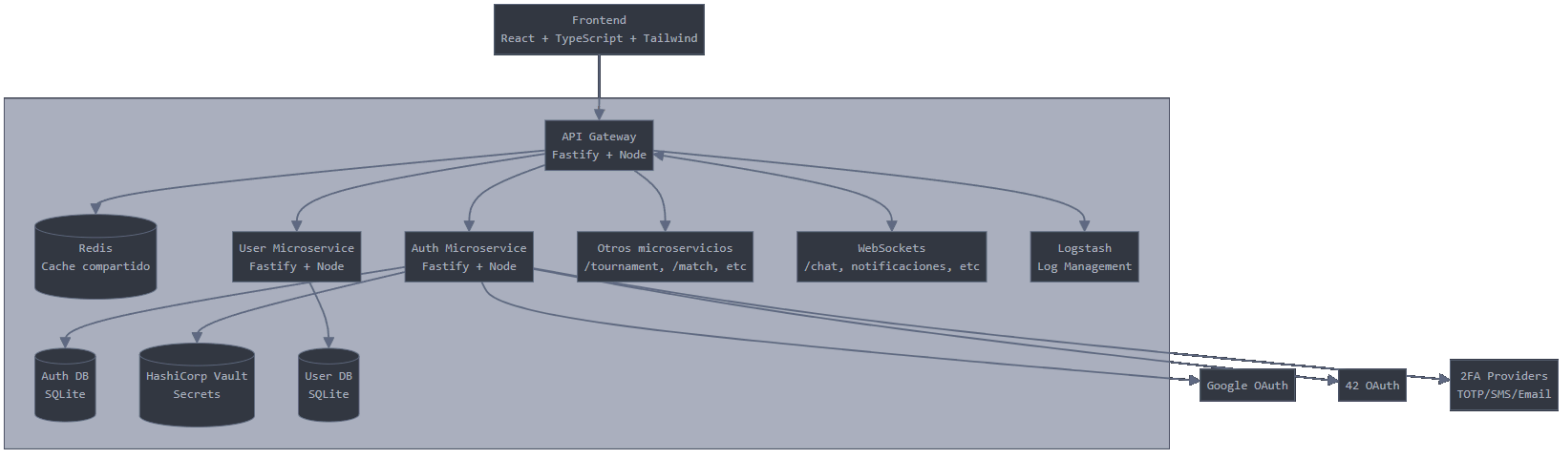
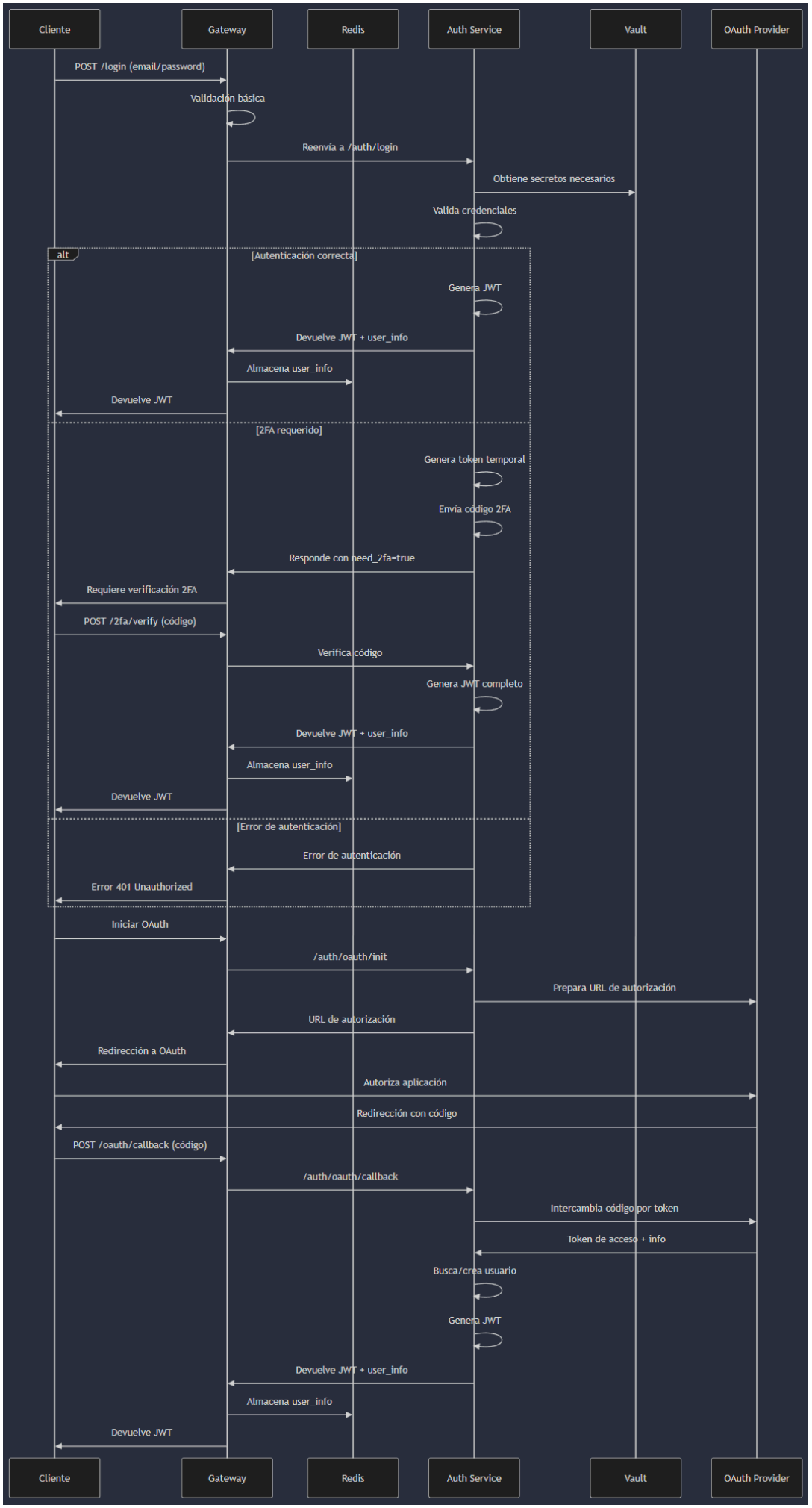
Planificación de Implementación: Sistema de Autenticación y Gestión de Usuarios

Basado en tu descripción y los requisitos del documento, voy a detallar un plan de implementación para tu sistema de microservicios con enfoque en la autenticación y gestión de usuarios.

Arquitectura del Sistema



Flujos de Autenticación



Componentes del Sistema

1. API Gateway

El Gateway es el punto de entrada único para todas las peticiones y tiene las siguientes responsabilidades:

- **Enrutamiento**: Dirigir las solicitudes al microservicio apropiado
- **Autenticación**: Validar JWT y gestionar la información de usuario en Redis
- **Seguridad**: Implementar WAF/ModSecurity como primera barrera de defensa
- **WebSockets**: Gestionar conexiones WebSockets para chat y notificaciones
- **Logging**: Asignar request_id a cada solicitud para seguimiento en Logstash
- **Validación básica**: Verificar formato y estructura de datos entrantes

Puntos de mejora respecto a tu planteamiento original:

- Implementar el WAF específicamente en el Gateway (ya mencionado)
- Agregar Hashicorp Vault para gestión de secretos
- Validación de formularios a varios niveles, no solo en Gateway

2. Microservicio de Autenticación (/auth)

Base de datos /auth

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    username TEXT UNIQUE,
    password_hash TEXT,           -- NULL para usuarios OAuth
    salt TEXT,                    -- NULL para usuarios OAuth
    account_type ENUM('local', 'google', '42') NOT NULL,
    oauth_id TEXT,                -- ID del proveedor OAuth
    has_2fa BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    is_deleted BOOLEAN DEFAULT FALSE
);

CREATE TABLE two_factor (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    type ENUM('email', 'sms', 'app', 'qr') NOT NULL,
    secret TEXT,                  -- Para apps TOTP
    phone_number TEXT,            -- Para SMS
    backup_codes TEXT,            -- Códigos de respaldo
    is_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

```
CREATE TABLE sessions (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    token_hash TEXT NOT NULL,      -- Hash del token (no el token
completo)
    ip_address TEXT,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP NOT NULL,
    revoked BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE password_reset (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    token TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP NOT NULL,
    is_used BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Endpoints de /auth

```
# Registro y autenticación básica
POST /auth/register                - Registrar nuevo usuario
POST /auth/login                   - Login con email/password
POST /auth/logout                  - Logout (revoca token)
GET  /auth/validate                - Validar token JWT
POST /auth/refresh                 - Refrescar token JWT

# Gestión de contraseñas
POST /auth/password/reset-request  - Solicitar reset de contraseña
POST /auth/password/reset          - Resetear contraseña con token

# OAuth
GET  /auth/oauth/:provider/init    - Iniciar flujo OAuth (google, 42)
POST /auth/oauth/:provider/callback - Callback de OAuth

# 2FA
POST /auth/2fa/enable              - Habilitar 2FA
POST /auth/2fa/verify              - Verificar código 2FA
GET  /auth/2fa/setup               - Configuración de 2FA (QR, secreto)
POST /auth/2fa/backup-codes        - Generar códigos de respaldo
POST /auth/2fa/disable             - Deshabilitar 2FA

# GDPR y cuenta
GET  /auth/me                      - Información de la cuenta actual
DELETE /auth/user                  - Eliminar cuenta
```

Objeto user_info (Redis)

```
{
  "user_id": "12345",
  "email": "usuario@ejemplo.com",
  "username": "usuario123",
  "display_name": "Jugador Pro",  // Del perfil de usuario
  "account_type": "local",        // "local", "google", "42"
  "has_2fa": true,
  "is_active": true,
  "last_login": "2025-03-22T10:30:00Z",
  "avatar_url": "/avatars/default.png",  // URL relativa
  "created_at": "2025-01-15T00:00:00Z"
}
```

3. Microservicio de Usuario (/user)

Base de datos /user

```
CREATE TABLE profiles (
    id INTEGER PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL,  -- Vinculado con auth.users.id
    display_name TEXT UNIQUE,
    bio TEXT,
    avatar_url TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_online BOOLEAN DEFAULT FALSE,
    last_online TIMESTAMP,
    is_anonymized BOOLEAN DEFAULT FALSE
);

CREATE TABLE friends (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    friend_id INTEGER NOT NULL,
    status ENUM('pending', 'accepted', 'rejected', 'blocked') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES profiles(user_id),
    FOREIGN KEY (friend_id) REFERENCES profiles(user_id),
    UNIQUE(user_id, friend_id)
);
```

```
CREATE TABLE statistics (
    id INTEGER PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL,
    wins INTEGER DEFAULT 0,
    losses INTEGER DEFAULT 0,
    tournaments_joined INTEGER DEFAULT 0,
    tournaments_won INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES profiles(user_id)
);

CREATE TABLE gdpr_requests (
    id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    request_type ENUM('export', 'delete', 'anonymize') NOT NULL,
    status ENUM('pending', 'processing', 'completed', 'failed') NOT
NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES profiles(user_id)
);
```

Endpoints de /user

```
# Perfil
GET    /user/profile                - Obtener perfil del usuario
actual
PUT    /user/profile                - Actualizar perfil
GET    /user/profile/:username      - Ver perfil de otro usuario
POST   /user/avatar                 - Subir avatar
DELETE /user/avatar                 - Eliminar avatar

# Amigos
GET    /user/friends                - Listar amigos
POST   /user/friends/:id            - Enviar solicitud de amistad
PUT    /user/friends/:id            - Aceptar/rechazar solicitud
DELETE /user/friends/:id            - Eliminar amigo o bloquear

# Estadísticas
GET    /user/stats                  - Estadísticas del usuario
GET    /user/matches                - Historial de partidas

# GDPR
GET    /user/gdpr/export            - Exportar datos (GDPR)
POST   /user/gdpr/anonymize         - Anonimizar datos (GDPR)
GET    /user/gdpr/request-status/:id - Estado de solicitud GDPR
```

Estrategias y Recomendaciones

Implementación de Seguridad

1. **JWT (JSON Web Tokens)**
    - Estructura recomendada:

```
{
  "iss": "pong-platform",        // Emisor
  "sub": "12345",                // ID del usuario
  "exp": 1719231000,             // Tiempo de expiración (15-30 min)
  "iat": 1719229200,             // Tiempo de emisión
  "jti": "unique-jwt-id-123"     // ID único del token
}
```

    - Usar algoritmo RS256 (asimétrico)
    - Almacenar claves privadas en HashiCorp Vault
    - Tiempo de expiración corto (15-30 minutos)
    - Implementar token de refresco para sesiones largas
2. **2FA (Autenticación de Dos Factores)**
    - Implementar múltiples métodos:
        - TOTP (Google Authenticator, Authy)
        - Códigos por SMS (considerando costos)
        - Códigos por email (más económico)
        - Códigos de respaldo (para emergencias)
    - Flujo de verificación separado del login principal
    - Opción para omitir en dispositivos confiables (con cookie específica)
3. **HashiCorp Vault**
    - Almacenar:
        - Claves privadas JWT
        - Credenciales OAuth (client_id, client_secret)
        - Credenciales de base de datos
        - Claves API para servicios externos (SMS, email)
    - Integrar con sistema de rotación de claves
4. **WAF/ModSecurity**
    - Configurar en el Gateway
    - Reglas para prevenir:
        - Inyección SQL
        - XSS (Cross-Site Scripting)
        - CSRF (Cross-Site Request Forgery)
        - Ataques de fuerza bruta

Gestión de Datos y Validación

1. **Estrategia de validación en capas:**
    - **Frontend**: Validación inmediata para UX
    - **Gateway**: Validación estructural, tamaño, formato
    - **Microservicios**: Validación específica de negocio
2. **Manejo de duplicados:**
    - Emails: Deben ser únicos globalmente
    - Usernames: Únicos en auth para login
    - Display names: Únicos en user para mostrar en torneos
    - Solución: Validar duplicados antes de crear, ofrecer alternativas
3. **Uso eficiente de Redis:**
    - Clave: `user:{id}:info`
    - TTL: Levemente mayor que expiración JWT (ej: JWT 15min, Redis 20min)
    - Actualizar en cada login exitoso y cambio de perfil
    - Eliminar en logout explícito

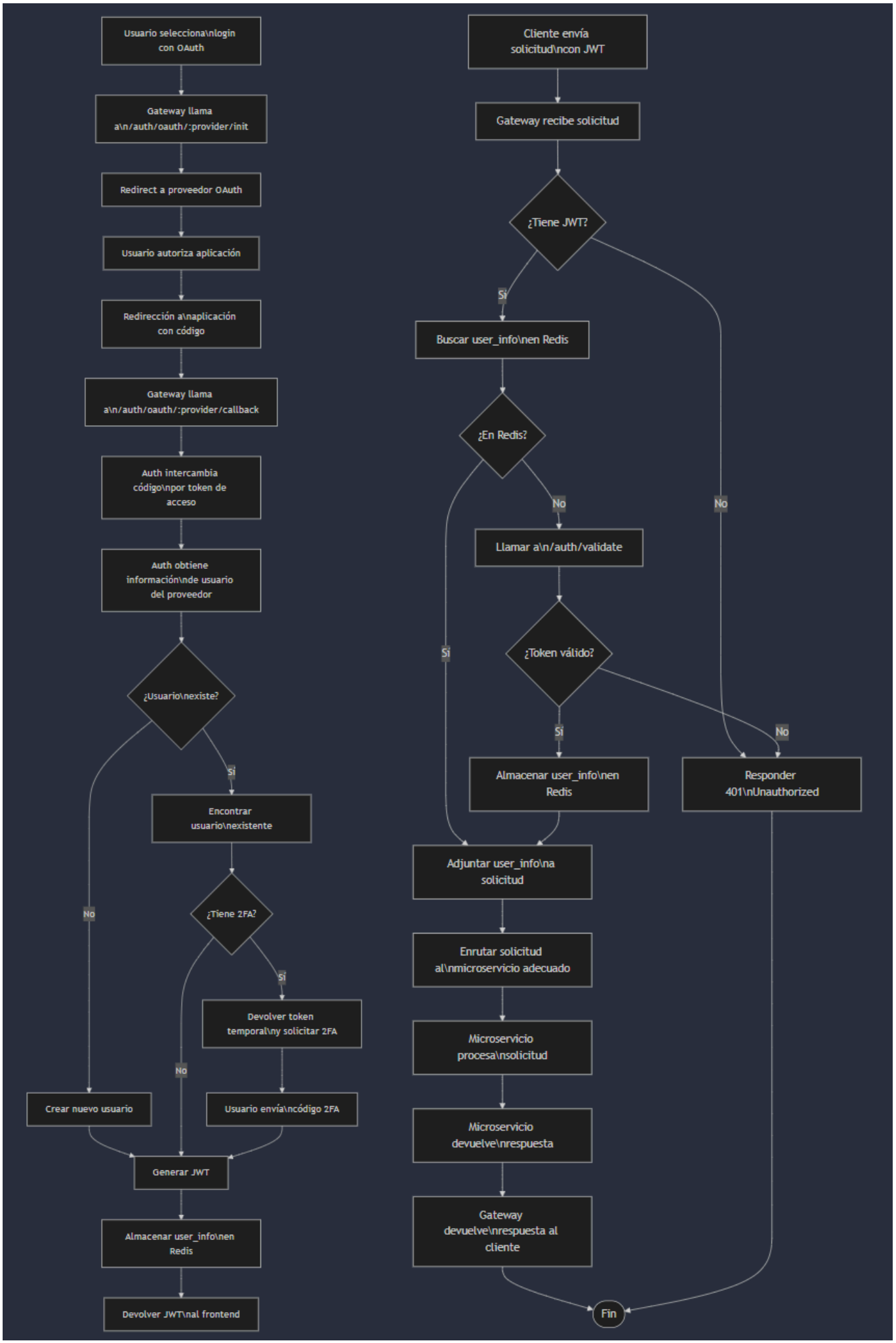
GDPR y Gestión de Usuarios

1. **Operaciones de cuenta:**
    - **Eliminación:**
        - Soft delete en auth.users (is_deleted = true)
        - Anonimizar datos en user.profiles
        - Eliminar datos sensibles (email, nombre real)
        - Mantener estadísticas agregadas
    - **Anonimización:**
        - Reemplazar datos personales con valores genéricos
        - Preservar estadísticas y partidas para integridad
        - Marcar cuenta como anonimizada
    - **Exportación de datos:**
        - Recopilar datos de todos los microservicios
        - Formato JSON estructurado
        - Incluir todos los datos personales
2. **Manejo de errores:**
    - Formato estándar de error:

```
{
  "error": true,
  "code": "AUTH_INVALID_CREDENTIALS",
  "message": "Credenciales inválidas",
  "request_id": "abc123"
}
```

    - Agrupar errores por categoría (auth, validation, server)
    - Incluir siempre request_id para correlación con logs

Diagramas de Flujo para Casos Específicos



Consideraciones adicionales

1. **Manejo de sesiones múltiples:**
    - Permitir múltiples dispositivos conectados
    - Opción para ver y cerrar sesiones activas
    - Detectar y notificar inicios de sesión inusuales
2. **Resistencia a fallos:**
    - Si Redis falla: Validar directamente con /auth
    - Si Auth falla: Denegar acceso temporalmente
    - Implementar reintentos con backoff exponencial
3. **Monitoreo y logging:**
    - Log estructurado con campos consistentes
    - Correlacionar logs mediante request_id
    - Alertas para intentos de autenticación sospechosos
    - Métricas de uso y rendimiento
4. **Optimizaciones:**
    - Caché de perfiles populares
    - Conexiones persistentes entre microservicios
    - Compresión de respuestas HTTP

¿Qué podrías mejorar en tu planteamiento original?

1. **Separación más clara de responsabilidades:** Tu propuesta ya tiene una buena separación, pero podría refinarse más la distinción entre autenticación (/auth) y gestión de usuarios (/user).
2. **Seguridad reforzada:** Implementar explícitamente HashiCorp Vault para secretos y configurar WAF/ModSecurity como indican los requisitos.
3. **GDPR explícito:** Asegurar que las funcionalidades GDPR estén bien definidas e implementadas según los requisitos.
4. **Validación en capas:** Implementar validación no solo en el Gateway sino también en microservicios para tener defensas en profundidad.
5. **Mayor detalle en 2FA:** Detallar los métodos específicos de 2FA y su implementación.