TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

DOCUMENTACIÓN DE LA API REST

Autor:

D. Marc Hernández Montesinos

Directora:

Dra. Dña. *Angélica Guzmán Ponce*

Murcia, Junio de 2026

TRABAJO FIN DE GRADO

UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

DOCUMENTACIÓN DE LA API REST

Autor:

D. Marc Hernández Montesinos

Directora:

Dra. Dña. *Angélica Guzmán Ponce*

Murcia, Junio de 2026

https://tfg.marchernandez.es/videos/video-demostrativo.mp4

ÍNDICE

DOCUMENTACIÓN DE LA API REST

Endpoints SSO + PKI, autenticación y ejemplos curl

Documento separado del Trabajo de Fin de Grado *"Diseño e implementación de un sistema integrado de PKI y SSO para organizaciones pequeñas"* de Marc Hernández Montesinos (UCAM, Grado en Ingeniería Informática).

Campo	Valor
Documento	Documentación de la API REST
Versión	1.0
Fecha	1 junio 2026

Campo	Valor
URL pública	https://tfg.marchernandez.es/manuales/Manual_API_REST.pdf
TFG asociado	https://tfg.marchernandez.es

Este manual conserva la numeración interna original (sección D.x) por trazabilidad con las versiones previas del documento. Las referencias cruzadas que apuntan al cuerpo del TFG (capítulos 1-10, anexos A-D) se mantienen tal cual y son válidas frente al PDF principal del TFG.

Este anexo documenta los endpoints HTTP expuestos por el sistema, agrupados en tres bloques: la **API REST OAuth 2.0/OIDC del SSO** (interfaz máquina-a-máquina, JSON), los **endpoints web del portal PKI** (interfaz humano-a-máquina, formularios POST con CSRF) y los **endpoints públicos sin autenticación** (OCSP, CRL, repositorio CA). Se incluyen verbo HTTP, ruta, mecanismo de autenticación, parámetros, formato de respuesta y códigos de estado, con ejemplos `curl` reales.

D.1 Visión global

D.1.1 Tres tipos de interfaz HTTP

El sistema **no expone una única API REST monolítica**. Distingue tres familias de endpoints, cada una con su propio estilo de interacción, autenticación y formato de respuesta:

Familia	Estilo	Cliente típico	Formato
API SSO (`sso.marchernandez.es/api/*` + `/.well-known/*`)	REST programática (JSON)	Aplicaciones cliente OAuth/OIDC	`application/json`
Portal PKI (`pki.marchernandez.es/*`)	Páginas web con formularios POST + CSRF	Navegador del usuario o admin	`text/html`, descarga `applicatio`

Familia	Estilo	Cliente típico	Formato
			`n/x-pem-file`
Endpoints públicos PKI (`ocsp/crl/ca.marchernandez.es`)	RPC binario o descarga estática	Software cliente (OpenSSL, navegador, librerías PKIX)	`applicatio n/ocsp-response`, `applicatio n/pkix-crl`, `applicatio n/x-pem-file`

Honestidad técnica: llamar "API REST" a la totalidad del sistema sería incorrecto. La interfaz del SSO es REST en el sentido estándar; el portal PKI es una aplicación web tradicional cuya "API HTTP" son sus rutas PHP, y los endpoints OCSP/CRL/CA son protocolos binarios o descargas estáticas. Este anexo respeta esa distinción.

D.1.2 Tabla resumen de endpoints

Método	Ruta	Autenticación	Finalidad
SSO-REST			

Método	Ruta	Autenticación	Finalidad
programática			
GET	`/.well-known/openid-configuration`	Pública	OIDC Discovery (RFC 8414): publica endpoints, algoritmos, scopes
GET	`/api/jwks.php`	Pública (cacheable)	Publica claves públicas RS256 para validar JWT (RFC 7517)
GET	`/api/authorize.php`	Cookie de sesión, mTLS o login interactivo	OAuth 2.0 Authorization Endpoint (RFC 6749 Sección 3.1) + PKCE (RFC 7636)
POST	`/api/token.php`	`client_id` + `client_secret`	OAuth 2.0 Token Endpoint (RFC 6749 Sección 3.2) -

Método	Ruta	Autenticación	Finalidad
			`authorization_code` y `refresh_token`
`GET`	`/api/userinfo.php`	`Authorization: Bearer <access_token>`	OIDC UserInfo Endpoint (Core Sección 5.3)
`POST`	`/api/verify.php`	Pública (rate-limited)	Validación de token *off-band* (uso interno; complemento a UserInfo)
`GET/POST`	`/api/logout.php`	Cookie de sesión (opcional)	OIDC RP-Initiated Logout
PKI-Portal			

Método	Ruta	Autenticación	Finalidad
web (autenticado)			
`GET/POST`	`/login.php`	Pública (entrada al flujo)	Login local o redirección a SSO
`GET`	`/api/auth/callback.php`	Sesión PHP + `state`	Callback OAuth2 desde el SSO

Método	Ruta	Autenticación	Finalidad
GET/POST	/request.php	Sesión PHP autenticada + CSRF	Solicitud de certificado (formulario)
GET	/certificates.php	Sesión PHP autenticada	Listado de certificados del usuario
GET	/certificate-detail.php?id=N	Sesión PHP autenticada (propietario)	Detalles de un certificado
GET	/certificate-download.php?id=N&format=pem|der	Sesión PHP autenticada (propietario)	Descarga del certificado emitido
GET/POST	/revoke.php?id=N	Sesión PHP autenticada (propietario) + CSRF + confirmación textual	Revocación del propio certificado

Método	Ruta	Autenticación	Finalidad
GET/POST	`/admin/request-review.php?id=N`	Sesión + rol `admin`/`superadmin` + CSRF	Aprobar/rechazar/emitir solicitud
GET/POST	`/admin/certificates.php`	Sesión + rol admin	Gestión global de certificados (revocar como admin)
GET/POST	`/admin/ca.php`	Sesión + rol `superadmin`	Crear / gestionar CAs intermedias
GET	`/admin/crl.php?ca_id=N`	Sesión + rol admin	Generación manual de CRL

Método	Ruta	Autenticación	Finalidad
`GET`	`/admin/audit.php`	Sesión + rol admin	Consulta del log de auditoría PKI
Endpoints públicos PKI			
`GET`	`http://ca.marchernandez.es/`	Pública	Página informativa con CAs disponibles

Método	Ruta	Autenticación	Finalidad
GET	`http://ca.marchernandez.es/{root|intermediate}.{crt|der|pem}`	Pública	Descarga directa por nombre canónico
GET	`http://ca.marchernandez.es/chain.pem`	Pública	Cadena completa de CAs activas
GET	`http://ca.marchernandez.es/download.php?ca_id=N&format=pem|der`	Pública	Descarga por ID de CA
GET	`http://crl.marchernandez.es/`	Pública	Página informativa con CRLs disponibles
GET	`http://crl.marchernandez.es/{slug}.crl`	Pública	CRL en formato DER (consumida por AIA/CDP)
GET	`http://crl.marchernandez.es/download.php?ca_id=N&format=der|pem`	Pública	Descarga por ID de CA
POST	`http://ocsp.marchernandez.es/`	Pública	Consulta OCSP (RFC 6960) - `application/ocsp-request`
GET	`http://ocsp.marchernandez.es/{base64-request}`	Pública	Consulta OCSP por GET (request en base64url)

Nota sobre los puertos. La API SSO y el portal PKI solo se sirven por **HTTPS (443)**. Los endpoints CRL/OCSP/CA se sirven por **HTTP (80)** porque así lo exige

el ecosistema PKIX: el cliente que valida el certificado todavía no ha resuelto si confía en la cadena TLS, por lo que envolver la respuesta OCSP/CRL en TLS sería un *deadlock* circular. Las respuestas OCSP y las CRL están **firmadas criptográficamente** por la propia CA, lo que sustituye la garantía de integridad que TLS daría en otro contexto.

D.2 Convenciones comunes

D.2.1 HTTPS y certificado

- **API SSO y portal PKI:** HTTPS obligatorio. Cualquier petición HTTP se redirige con `301` a HTTPS por `.htaccess` (fragmentos relevantes en Manual de Código Relevante (Sección A.7)).
- **HSTS:** se sirve `Strict-Transport-Security: max-age=31536000; includeSubDomains; preload` en todos los endpoints HTTPS.
- **TLS 1.2 mínimo** (TLS 1.3 preferido). Cifradores configurados según Mozilla *Intermediate Compatibility*.

D.2.2 Codificación y formato

- Codificación de caracteres: **UTF-8** universal.
- Cuerpos de respuesta:
  - API SSO → `application/json; charset=utf-8` (con `JSON_UNESCAPED_UNICODE`).
  - Portal PKI → `text/html; charset=utf-8` o descarga PEM/DER según el endpoint.
  - OCSP → `application/ocsp-response` (binario DER).
  - CRL → `application/pkix-crl` (DER) o `application/x-pem-file` (PEM).
- Cuerpos de petición aceptados por `/api/token.php`:
  - `application/x-www-form-urlencoded` (formato canónico OAuth 2.0).

- `application/json` (alternativa aceptada por compatibilidad con clientes modernos).

D.2.3 Autenticación según endpoint

Mecanismo	Dónde se usa
Pública (sin autenticación)	`/.well-known/*`, `/api/jwks.php`, CRL/OCSP/CA repo, página de login del SSO o PKI
Sesión existente (cookie `mhsso_token`)	`/api/authorize.php` cuando el usuario ya está logueado en el SSO
mTLS (Apache valida certificado cliente)	`/api/authorize.php` cuando se autentica con FNMT/DNIe
`client_id` + `client_secret`	`/api/token.php` (en body o `Authorization: Basic`)
`Authorization: Bearer <access_token>` (JWT RS256)	`/api/userinfo.php`
Sesión PHP autenticada + CSRF token	Todo el portal PKI (no sólo GET-safe)

D.2.4 Códigos HTTP utilizados

Código	Significado	Cuándo se devuelve
`200`	OK	Operación exitosa
`204`	No Content	Pre-flight CORS (`OPTIONS`)

Código	Significado	Cuándo se devuelve
`301`	Redirección permanente	HTTP → HTTPS
`302`	Redirección temporal	OAuth2 redirect a `redirect_uri`, OIDC RP-Initiated Logout, `View::redirect()`
`400`	Bad Request	Parámetros OAuth inválidos, `redirect_uri` no autorizada, JSON malformado, CSRF inválido
`401`	Unauthorized	Token JWT inválido o ausente, `client_secret` incorrecto, `refresh_token` expirado
`403`	Forbidden	Acceso a admin sin rol adecuado, acceso a recurso ajeno (`certificate-detail`)
`404`	Not Found	Recurso inexistente (CA, CRL, certificado, solicitud)
`405`	Method Not Allowed	`GET` en endpoints que sólo aceptan `POST` y viceversa; cabecera `Allow:` presente
`415`	Unsupported Media Type	`Content-Type` distinto de `application/json` o `application/x-www-form-urlencoded` en `/api/token.php`
`429`	Too Many Requests	Rate limiting por IP (login, token, OCSP)
`500`	Internal Server Error	Excepción no controlada (logueada; nunca se devuelve traza al cliente)
`503`	Service Unavailable	Bootstrap del PKI no localizable (en `ocsp/index.php`, `crl/index.php`, `ca/index.php`)

D.2.5 Formato estándar de error (API SSO)

Las respuestas de error de la API SSO siguen la convención OAuth 2.0 (RFC 6749 Sección 5.2) extendida con `error_description` para facilitar diagnóstico:

```
{
  "error": "invalid_grant",
  "error_description": "El authorization_code ha expirado o ya fue
utilizado."
}
```

Códigos de error reconocidos:

- `invalid_request` - parámetros ausentes o malformados.
- `invalid_client` - `client_id`/`client_secret` no válidos.
- `invalid_grant` - code/refresh_token caducado, inválido o ya consumido; PKCE no superado.
- `unauthorized_client` - el cliente no tiene permitido este `grant_type`.
- `unsupported_grant_type` - `grant_type` no soportado.
- `unsupported_media_type` - `Content-Type` no soportado.
- `access_denied` - usuario no autorizado para la aplicación o certificado rechazado.
- `temporarily_unavailable` - *rate limit* superado u otro recurso transitoriamente saturado.
- `rate_limit_exceeded` - alias específico para 429 en endpoints internos.
- `invalid_token` - Bearer token ausente, malformado, expirado o revocado.
- `method_not_allowed` - verbo HTTP no permitido.

D.3 API SSO (`sso.marchernandez.es`)

D.3.1 `GET /.well-known/openid-configuration` - OpenID Connect Discovery

Finalidad. Publica los *metadatos* del proveedor OIDC para que los *Relying Parties* descubran automáticamente endpoints, algoritmos, scopes y *claims*

soportados sin configuración manual. La URL es extension-less por exigencia de la OIDC Core 1.0 Sección 4 y se sirve mediante una `RewriteRule` que mapea internamente a `openid-configuration.php`.

Autenticación. Pública. Cacheable.

Respuesta (`200 OK`, `application/json`). Documento JSON con los siguientes campos:

```
{
  "issuer": "https://sso.marchernandez.es",
  "authorization_endpoint": "https://sso.marchernandez.es/api/authorize.php",
  "token_endpoint": "https://sso.marchernandez.es/api/token.php",
  "userinfo_endpoint": "https://sso.marchernandez.es/api/userinfo.php",
  "jwks_uri": "https://sso.marchernandez.es/api/jwks.php",
  "response_types_supported": ["code"],
  "subject_types_supported": ["public"],
  "id_token_signing_alg_values_supported": ["RS256"],
  "scopes_supported": ["openid", "profile", "email"],
  "token_endpoint_auth_methods_supported": ["client_secret_post",
"client_secret_basic"],
  "claims_supported": ["sub", "name", "preferred_username", "email"]
}
```

Notas de honestidad. El documento actual no anuncia `code_challenge_methods_supported` ni `end_session_endpoint`, aunque ambos están soportados:

- PKCE S256 está implementado y es obligatorio cuando el cliente envía `code_challenge` (véase Sección D.3.3).
- El `/api/logout.php` implementa OIDC RP-Initiated Logout completo.

Publicar estos campos en el *discovery document* es una mejora trivial (un `INSERT` en el JSON) y figura como deuda menor; la funcionalidad está presente desde las migraciones 011 y 012.

D.3.2 `GET /api/jwks.php` - JSON Web Key Set

Finalidad. Publica las claves públicas RS256 en formato JWK (RFC 7517) para que los *Relying Parties* validen la firma de los `id_token` emitidos por el SSO.

Autenticación. Pública.

Cache. `Cache-Control: public, max-age=3600` (1 hora). Esto permite a los RPs cachear las claves entre rotaciones, pero también obliga a respetar una ventana de transición cuando se gira el `kid`.

Respuesta (`200 OK`). JSON con la estructura:

```
{
  "keys": [
    {
      "kty": "RSA",
      "use": "sig",
      "alg": "RS256",
      "kid": "20260217-001",
      "n": "0v...(módulo base64url)...fQ",
      "e": "AQAB"
    }
  ]
}
```

Cada par `kid` ↔ clave pública corresponde a una fila activa de la tabla `sso_keypairs` (véase Anexo E Sección E.2.6). La aplicación puede albergar varios `kid` activos simultáneamente durante una rotación; cada JWT emitido lleva en su header el `kid` que lo firmó.

D.3.3 `GET /api/authorize.php` - OAuth 2.0 Authorization Endpoint

Finalidad. Punto de entrada del flujo *Authorization Code* (RFC 6749 Sección 4.1) extendido con OIDC y PKCE (RFC 7636). Identifica al usuario y, si todo es correcto, redirige al `redirect_uri` con un `authorization_code` de un solo uso (TTL 60 s).

Autenticación. Tres caminos posibles, evaluados en orden:

1. **Sesión existente:** cookie `mhsso_token` válida → emite código inmediatamente.
2. **mTLS:** si Apache marca `SSL_CLIENT_VERIFY=SUCCESS` (certificado FNMT/DNIe/Omnipresence presentado) → autentica vía `CertificateAuth` y emite código con `cert_serial` en el JWT.
3. **Interactivo:** sin sesión ni certificado → redirige a `/login.php` propagando los parámetros OAuth.

Parámetros (query string).

Parámetro	Obligatorio	Descripción
`client_id`	Sí	Identificador público del cliente (`sso_applications.client_id`)
`redirect_uri`	Sí	URI exacta autorizada en `sso_applications.redirect_uri`
`response_type`	Sí	Debe ser `code`
`scope`	Recomendado	Por defecto `openid profile email`
`state`	Recomendado	Valor *anti-CSRF* opaco; se devuelve sin modificar
`code_challenge`	Opcional (recom.)	BASE64URL(SHA256(verifier)); si presente, exige PKCE en `/token`
`code_challenge_method`	Junto con anterior	Solo `S256`; `plain` se rechaza

Comportamiento.

- Si `client_id` o `redirect_uri` no son válidos → 400 JSON (no se redirige, RFC 6749 Sección 4.1.2.1: prohibido reflejar errores en una URI no confiable).
- Si la aplicación está en modo `access_mode='restricted'` y el usuario no figura en `sso_application_users` → redirige al login con `access_denied=1`.
- Si todo correcto → `302 Location: {redirect_uri}?code=XXX&state=YYY`.

Errores específicos.

Caso	Acción
`redirect_uri` no coincide	`400 invalid_request`, sin redirección

Caso	Acción
`response_type ≠ code`	Redirige con `error=unsupported_response_type`
`code_challenge_method=plain`	`400 invalid_request`
Certificado presentado, pero EKU bloqueado	Redirige con `error=access_denied` + descripción
Rate limit superado en `cert_authorize`	Redirige con `error=temporarily_unavailable`

D.3.4 `POST /api/token.php` - OAuth 2.0 Token Endpoint

Finalidad. Canjea el `authorization_code` (o un `refresh_token`) por un *access token* + *refresh token* + *id token*.

Autenticación. `client_id` + `client_secret` (verificado contra `client_secret_hash` con Argon2id). Admite dos vías:

- `Authorization: Basic base64(client_id:client_secret)` (RFC 6749 Sección 2.3.1).
- `client_id` y `client_secret` en el body (`client_secret_post`, también RFC 6749 Sección 2.3.1).

Petición (`application/x-www-form-urlencoded` o `application/json`).

```
grant_type=authorization_code
&code=AbCdEf123...
&client_id=mhsso_pki_client
&client_secret=...
&redirect_uri=https://pki.marchernandez.es/api/auth/callback.php
&code_verifier=...          # obligatorio si se envió code_challenge
```

O bien:

```
grant_type=refresh_token
&refresh_token=...
```

Respuesta (`200 OK`).

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjIw...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "ABab...(32 bytes base64url)...XY",
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6Ijw...",
  "scope": "openid profile email"
}
```

El `id_token` es un JWT RS256 con:

- `iss = https://sso.marchernandez.es`
- `aud = <client_id>`
- `sub = <user_id UUID>`
- `iat`, `exp`, `nbf` estándar.
- `nonce` si el cliente lo envió en `/authorize`.
- `cert_serial` (claim personalizado, solo si el login fue por certificado).
- `auth_method` (claim personalizado: `password`, `totp`, `certificate`, `smartcard`).

Errores.

Caso	Código	`error`
`grant_type` ausente o no soportado	400	`unsupported_grant_type`
Parámetros obligatorios ausentes	400	`invalid_request`
`client_id`/`client_secret` no válidos	401	`invalid_client`
Code caducado, ya consumido, no coincide `redirect_uri`	400	`invalid_grant`

Caso	Código	`error`
PKCE: code emitido con `code_challenge` y `code_verifier` ausente	400	`invalid_grant`
PKCE: `BASE64URL(SHA256(verifier)) ≠ code_challenge`	400	`invalid_grant`
`refresh_token` inválido o caducado	401	`invalid_grant`
Rate limit superado	429	`rate_limit_exceeded`
`Content-Type` no soportado	415	`unsupported_media_type`

Rate limit aplicado. 30 peticiones / 60 s por IP (`api_token`).

D.3.5 `GET /api/userinfo.php` - OIDC UserInfo Endpoint

Finalidad. Devuelve los *claims* del usuario asociados al *access token* presentado. Equivale al endpoint definido en OIDC Core 1.0 Sección 5.3.

Autenticación. Cabecera `Authorization: Bearer <access_token>`.

Respuesta (`200 OK`).

```
{
  "sub": "550e8400-e29b-41d4-a716-446655440000",
  "username": "marc",
  "email": "correo@marchernandez.es",
  "display_name": "Marc Hernández Montesinos",
  "role": "admin",
  "totp_enabled": true,
  "has_certificate": false,
  "has_smartcard": false,
  "last_login_at": "2026-05-12 13:08:42",
  "created_at": "2026-02-06 17:43:18"
}
```

Errores.

Caso	Código	`error`
Cabecera `Authorization` ausente	401	`invalid_token`
Token inválido o expirado	401	`invalid_token`
Verbo distinto de `GET`	405	`method_not_allowed`
Rate limit superado	429	`rate_limit_exceeded`

Rate limit aplicado. 60 peticiones / 60 s por IP (`api_userinfo`).

D.3.6 `POST /api/verify.php` - verificación off-band de token

Finalidad. Permite a aplicaciones internas validar un JWT sin pasar por `/userinfo`. No forma parte del estándar OIDC; es un atajo del propio SSO para integraciones que sólo necesitan saber "¿este token sigue vivo?". Las aplicaciones externas deberían usar `/userinfo` o validar el JWT localmente contra `/api/jwks.php`.

Petición (`application/json`).

```
{ "token": "eyJhbGciOiJSUzI1NiIs..." }
```

Respuesta (`200 OK`).

- Si válido:

```
{
  "valid": true,
  "user_id": "550e8400-...",
  "username": "marc",
  "email": "correo@marchernandez.es",
  "role": "admin",
  "auth_method": "totp"
}
```

- Si no válido:

```
{ "valid": false, "error": "Token inválido o expirado." }
```

Rate limit aplicado. 60 peticiones / 60 s por IP (`api_verify`).

Nota: este endpoint siempre devuelve 200 con `valid:false` cuando el token es inválido (no 401), porque la operación es una verificación, no una autenticación.

D.3.7 `GET` / `POST` `/api/logout.php` - OIDC RP-Initiated Logout

Finalidad. Implementa la sección OIDC *RP-Initiated Logout* (borrador final, recomendado por la *Logout Working Group*). El RP envía al usuario a este endpoint con `id_token_hint` (o `client_id`); el SSO destruye la sesión y, si la URI de redirección está autorizada, redirige al usuario a `post_logout_redirect_uri`.

Parámetros.

Parámetro	Obligatorio	Descripción
`id_token_hint`	Recomendado	JWT id_token emitido previamente para identificar el cliente
`client_id`	Alternativa o complemento	Si no hay id_token_hint, identifica el cliente explícitamente
`post_logout_redirect_uri`	Opcional	URI a redirigir tras logout; debe estar en `sso_applications.post_logout_redirect_uris`
`state`	Opcional	Eco devuelto en la URI final

Estrategia de identificación del cliente (en orden):

1. `id_token_hint` → `JWT::decode` → claim `aud`.
2. `client_id` explícito en la query.
3. **Inferencia** por `post_logout_redirect_uri`: si una sola aplicación tiene esta URI registrada, se identifica como cliente.

Si tras estos pasos sigue sin haber cliente identificable y se proporcionó `post_logout_redirect_uri`, se rechaza con `400 invalid_request` (mitigación de Open Redirect).

Comportamiento.

- Sesión SSO destruida (cookies `mhsso_token` y `mhsso_refresh` borradas, sesión PHP nativa destruida).
- Si hay `post_logout_redirect_uri` válida → redirige a ella (con `?state=YYY` si se envió).
- Si no se envió URI → redirige a `/login.php?logout=1`.

Errores.

Caso	Código	`error`
`post_logout_redirect_uri` presente, no se puede identificar cliente	400	`invalid_request`
`post_logout_redirect_uri` no autorizada para el cliente	400	`invalid_request`

D.4 Portal PKI (`pki.marchernandez.es`)

Naturaleza del portal. El portal PKI no es una API REST programática. Es una aplicación web tradicional servida en PHP con *server-side rendering*. Las "rutas" son ficheros `.php` cuyas respuestas son HTML; los formularios POST llevan token CSRF (verificado por `Auth::verifyCsrf()`) y devuelven HTML con redirecciones `302`. Se documenta aquí porque las rutas, parámetros y respuestas son estables y forman parte del contrato HTTP del sistema, pero no debe consumirse desde otras aplicaciones sin pasar por SSO + sesión PHP.

D.4.1 Tabla de endpoints del portal PKI

Método	Ruta	Rol mínimo	Acción
GET	`/login.php`	Anónimo	Formulario de login local o botón "Iniciar sesión con SSO"
POST	`/login.php`	Anónimo	Login local (usuario + contraseña + CSRF)
GET	`/api/auth/callback.php?code&state`	Sesión PHP + `oauth2_state` válido	Callback OAuth2 desde SSO; canjea code por token, crea sesión PKI
GET	`/logout.php`	Sesión PHP	Destruye sesión PKI local
GET	`/dashboard.php`	Sesión PHP	Panel principal
GET	`/certificates.php`	Sesión PHP	Listado de certificados propios
GET	`/certificate-detail.php?id=N`	Sesión PHP propietario	Vista detallada de un certificado

Método	Ruta	Rol mínimo	Acción
GET	/certificate-download.php?id=N&format=pem|der	Sesión PHP propietario	Descarga del certificado en PEM o DER
GET	/profile.php	Sesión PHP	Perfil y configuración
GET	/sessions.php	Sesión PHP	Sesiones activas del usuario en el portal
GET	/request.php	Sesión PHP	Formulario de solicitud de certificado
POST	/request.php	Sesión PHP + CSRF	Envía la solicitud (CSR adjunto o generada)
GET	/revoke.php	Sesión PHP	Página para revocar certificados propios
GET	/revoke.php?id=N	Sesión PHP propietario	Preselecciona un certificado a revocar
PO	/revoke.php	Sesión PHP +	Ejecuta la revocación

Método	Ruta	Rol mínimo	Acción
ST		CSRF + confirmación textual `REVOCAR`	
GET	`/admin/index.php`	`admin`/`superadmin`	Panel de administración
GET	`/admin/users.php`	admin	Gestión de usuarios
GET	`/admin/requests.php`	admin	Cola de solicitudes pendientes
GET	`/admin/request-review.php?id=N`	admin	Vista de revisión de una solicitud
POST	`/admin/request-review.php`	admin + CSRF	Acciones `approve`, `reject`, `issue`
GET	`/admin/certificates.php`	admin	Listado global de certificados

Método	Ruta	Rol mínimo	Acción
GET	`/admin/certificate-view.php?id=N`	admin	Vista detallada (incluye PEM completo) y opciones admin
GET	`/admin/ca.php`	`superadmin`	Gestión de CAs (raíz + intermedias)
POST	`/admin/ca.php`	`superadmin` + CSRF	Crear CA intermedia (formulario)
GET	`/admin/crl.php`	admin	Listado y generación manual de CRL por CA
GET	`/admin/crl.php?ca_id=N&action=generate`	admin	Genera CRL bajo demanda
GET	`/admin/audit.php`	admin	Consulta del log de auditoría con filtros

D.4.2 Detalle de los flujos clave

D.4.2.1 Solicitud de certificado (`POST /request.php`)

Petición (`application/x-www-form-urlencoded`).

- Campos del *Subject*: `common_name`, `organization`, `organizational_unit`, `locality`, `state`, `country`, `email`.
- *Subject Alternative Names*: `san_dns`, `san_ip`, `san_email`.

- Plantilla y CA: `template_id`, `ca_id` (solo intermedias en el desplegable).
- Tamaño de clave: `key_size` (2048/3072/4096).
- Validez solicitada: `validity_days` (sujeta a aprobación admin).
- `csr_pem`: opcional, si el usuario aporta su propia CSR. Si está vacío, la clave se genera en el navegador con Web Crypto API y la CSR se ensambla *client-side*; la clave privada nunca se envía al servidor.
- `csrf_token`: obligatorio (verificado contra `$_SESSION['csrf_token']`).

Respuesta: `200 text/html` con la página de confirmación. Si la solicitud incluyó generación local de clave, se muestra la clave privada en una `<textarea>` con instrucciones para descargarla (vía Blob/`URL.createObjectURL`); no queda copia en BD ni en logs.

D.4.2.2 Aprobación y emisión (`POST /admin/request-review.php`)

Tres acciones por valor de `action`:

`action`	Efecto
`approve`	Marca solicitud como `approved`. Permite ajustar `override_validity_days` (admin puede reducir la validez solicitada).
`reject`	Marca solicitud como `rejected`. Obliga a `reject_reason` no vacío.
`issue`	Firma el certificado mediante `PKIEngine::signCertificate`. Devuelve serial. Sólo aplicable si la solicitud está en estado `approved`.

Todas las acciones registran un evento en `audit_log` (Anexo E Sección E.3.8) y exigen `csrf_token`.

D.4.2.3 Revocación por el usuario (`POST /revoke.php`)

Para reducir errores accidentales, la revocación exige tres confirmaciones:

1. `csrf_token` válido.

2. El campo `cert_id` debe pertenecer al usuario y estar en estado `valid`.
3. El usuario debe escribir literalmente la palabra `REVOCAR` en el campo `confirmation` (case-sensitive).

Razones admitidas (`reason`): `unspecified`, `keyCompromise`, `affiliationChanged`, `superseded`, `cessationOfOperation`, `certificateHold`, `privilegeWithdrawn` (subset operativo de RFC 5280).

Tras revocar se invoca `PKIEngine::revokeCertificate`, que actualiza `certificates.status='revoked'`, escribe en `index.txt` de la CA y dispara la regeneración de la CRL afectada.

D.5 Endpoints públicos PKI (sin autenticación)

D.5.1 Repositorio CA (`http://ca.marchernandez.es`)

Finalidad. Exposición pública de los certificados raíz e intermedios para que los clientes puedan instalarlos en su *trust store* y verificar las cadenas X.509.

Rutas.

Ruta	Respuesta	Cache
`GET /`	Página HTML informativa con tabla de CAs activas	`max-age=86400`
`GET /root.crt`	Certificado Root CA en PEM (`application/x-pem-file`)	`max-age=86400`
`GET /root.der`	Certificado Root CA en DER (`application/x-x509-ca-cert`)	`max-age=86400`
`GET /intermediate.crt`	Intermedia "principal" en PEM	`max-age=86400`
`GET /intermediate.der`	Intermedia en DER	`max-age=86400`
`GET /chain.pem`	Concatenación PEM de todas las CAs activas (orden:	`max-age=86400`

Ruta	Respuesta	Cache
	intermedias primero, raíz al final)	
`GET /download.php?ca_id=N&format=pem|der`	Cualquier CA específica por ID	`max-age=86400`

Códigos de error. `404` si la CA solicitada no existe o está inactiva; `503` si el bootstrap del PKI falla.

D.5.2 Distribución CRL (`http://crl.marchernandez.es`)

Finalidad. Exposición pública de las Listas de Revocación. Las URLs se inyectan automáticamente en el campo `crlDistributionPoints` (CDP) de cada certificado emitido (Anexo F).

Rutas.

Ruta	Respuesta
`GET /`	Página HTML con CAs activas y enlaces a sus CRL
`GET /{slug}.crl`	CRL en DER (`application/pkix-crl`) - esta es la URL que figura en los certificados
`GET /download.php?ca_id=N&format=der|pem`	CRL específica por ID, con elección de formato

Comportamiento de regeneración bajo demanda. Si el fichero `{slug}.crl` no existe en disco (caso raro tras el cron de cada 4 h), `serveCRL()` busca la CA correspondiente y dispara `CRLManager::generate()` antes de servirlo, evitando un `404` durante una ventana corta tras un fallo del cron.

Auditoría. Cada descarga vía `/download.php` registra una fila en `crl_downloads` con IP y User-Agent (Anexo E Sección E.3.9). Las descargas vía URL directa `{slug}.crl` son anónimas (servidas por Apache estáticamente cuando el fichero existe).

Cache. `Cache-Control: public, max-age=3600` (1 h). Se eligen 3600 s deliberadamente más cortos que tanto la regeneración del repositorio (4 h, parámetro `crl_validity_hours = 4`) como el `nextUpdate` del DER firmado (24 h, por la granularidad de `openssl ca -gencrl -crldays`). Esto fuerza a un cliente respetuoso del HTTP a revalidar cada hora aunque el `nextUpdate` interno todavía no haya caducado, mejorando la frescura efectiva sin renunciar al margen estructural del DER (cf. Sección 7.8 del cuerpo y Sección 9.4.10 bis).

D.5.3 OCSP responder (`http://ocsp.marchernandez.es`)

Finalidad. Verificación en tiempo real del estado de un certificado emitido por cualquiera de las CAs (RFC 6960).

Métodos.

- `POST /` - método canónico. El body contiene la `OCSPRequest` en DER, con `Content-Type: application/ocsp-request`.
- `GET /{base64url(OCSPRequest)}` - método alternativo (RFC 6960 Sección A.1). Útil para clientes y firewalls que sólo permiten GET.

Respuesta.

- `Content-Type: application/ocsp-response`.
- Body: `OCSPResponse` en DER firmada por la propia CA (véase Manual de Código Relevante).
- `Cache-Control: public, max-age=<ocsp.cache_ttl>` (3600 s por defecto).

Códigos de respuesta OCSP (RFC 6960 Sección 4.2.1):

Status	Significado
`0` successful	Respuesta completada
`1` malform	Petición malformada (devuelta por `OCSPHandler::buildMalformedResponse()`)

Status	Significado
edRequ est	
2 internal Error	Error interno

Auditoría. Cada consulta se registra en `ocsp_queries` con `serial_number`, `response_status`, `response_time_ms` (Anexo E Sección E.3.7).

Límite de tamaño. Las peticiones POST superiores a `ocsp.max_request_size` (10240 bytes por defecto) se rechazan como malformadas.

D.6 Seguridad transversal de la API

D.6.1 PKCE (RFC 7636)

- Soportado en `/api/authorize.php` y exigido en `/api/token.php` si el código fue emitido con `code_challenge`.
- Método `S256` obligatorio: `code_challenge = BASE64URL(SHA256(code_verifier))`. El método `plain` se rechaza con `400 invalid_request`.
- El `code_verifier` debe tener entre 43 y 128 caracteres del alfabeto `A-Z a-z 0-9 - . _ ~` (RFC 7636 Sección 4.1).
- La verificación se hace en tiempo constante (`hash_equals`) para evitar *timing leaks*.
- Compatibilidad legada: si el `authorization_code` se emitió sin `code_challenge`, `/token` no exige `code_verifier`; esto permite migración gradual de clientes antiguos sin romper integraciones existentes. La cobertura PKCE se monitoriza desde `sso_audit_log` (Anexo E Sección E.2.11).

D.6.2 Validación de `redirect_uri`

- Comparación exact-match, sin *wildcards* ni *path-matching*. Cualquier discrepancia (mayúsculas, slash final, parámetro extra) se rechaza con `400 invalid_request`.
- Si `redirect_uri` es inválida, el SSO no redirige y devuelve `400 JSON` directamente (RFC 6749 Sección 4.1.2.1). Esto evita que un atacante use el SSO como *open redirect*.
- Equivalente para logout: `post_logout_redirect_uri` se valida contra `sso_applications.post_logout_redirect_uris` (lista exact-match, una por línea).

DI.6.3 Validación de `aud`, `iss`, `exp` en los JWT

- **`iss` (issuer):** siempre `https://sso.marchernandez.es`. El RP debe verificarlo.
- **`aud` (audience):** el `client_id` del RP. El RP debe rechazar tokens cuyo `aud` no coincida con su propio `client_id`.
- **`exp` (expiration):** TTL por defecto 3600 s para `access_token`; el RP debe rechazar tokens expirados (con tolerancia ≤ 60 s de *clock skew* recomendada).
- **`nbf` (not before):** presente y equivalente a `iat`.
- **`iat` (issued at):** marca temporal de emisión.
- **`kid` (en header):** el RP debe seleccionar la clave pública correspondiente en `/api/jwks.php`.

DII.6.4 Rate limiting

Implementado mediante `MaHerMoSSO\Security\RateLimiter` apoyado en la tabla `sso_rate_limits` (ventana deslizante por `(identifier, action)`).

Endpoint	Acción	Límite	Identificador
`/api/token.php`	`api_token`	30 / 60 s	IP
`/api/userinfo.php`	`api_userinfo`	60 / 60 s	IP
`/api/verify.php`	`api_verify`	60 / 60 s	IP
`/api/authorize.php` (rama mTLS)	`cert_authorize`	10 / 60 s	IP
`/login.php`	`login`	5 fallos / 15 min	username + IP
`/forgot-password.php`	`password_reset`	3 / 60 min	email + IP

Superar el límite devuelve `429 rate_limit_exceeded` y registra un evento `rate_limit_exceeded` en `sso_audit_log` (Anexo E Sección E.2.11).

D.6.5 CSRF en el portal PKI

- Todo formulario `POST` del portal incluye `<input type="hidden" name="csrf_token" value="...">`.
- `Auth::verifyCsrf()` compara con `hash_equals` contra `$_SESSION['csrf_token']`.
- Los endpoints REST del SSO no usan CSRF porque exigen `client_secret` o `Authorization: Bearer` (que ya autentican la petición por sí mismos) y porque devuelven JSON, no HTML mutante.
- `/api/logout.php` admite `GET` y `POST`: el `GET` es seguro porque las consecuencias destructivas se limitan a la propia sesión del usuario (y son lo deseado en RP-Initiated Logout).

D.6.6 Cabeceras de seguridad

Todos los endpoints del SSO y del portal PKI envían:

- `X-Content-Type-Options: nosniff`

- `X-Frame-Options: DENY` (los endpoints JSON, además, son inutilizables como *iframe*)
- `Strict-Transport-Security: max-age=31536000; includeSubDomains; preload`
- `Referrer-Policy: strict-origin-when-cross-origin`
- `Cache-Control: no-store` en endpoints sensibles (`/api/authorize.php`, `/api/token.php`, `/api/userinfo.php`, `/api/logout.php`).
- `Access-Control-Allow-Origin: *` solo en `/api/token.php`, `/api/userinfo.php`, `/api/jwks.php`, `/api/verify.php` (necesario para clientes SPA con backend separado). El resto **no** habilita CORS.

D.7 Ejemplos `curl` reales

Los ejemplos siguientes son reproducibles sobre el despliegue real, con la salvedad de que requieren un `client_id`/`client_secret` válidos. Los valores `mhsso_pki_client` y secretos son ilustrativos.

D.7.1 Discovery + JWKS

```
# 1. Descubrir endpoints
curl -s https://sso.marchernandez.es/.well-known/openid-configuration | jq .

# 2. Obtener claves públicas
curl -s https://sso.marchernandez.es/api/jwks.php | jq .keys[0]
```

D.7.2 Flujo `authorization_code` con PKCE (extremo a extremo)

```
# Paso 1: generar code_verifier (43-128 chars unreserved) y code_challenge
S256
VERIFIER=$(openssl rand -base64 32 | tr -d '=+/' | cut -c1-64)
CHALLENGE=$(printf '%s' "$VERIFIER" | openssl dgst -binary -sha256 | base64 |
tr -d '=+/' | tr '/+' '_-')

echo "code_verifier=$VERIFIER"
echo "code_challenge=$CHALLENGE"

# Paso 2: pedir authorization_code (en navegador, no curl, porque exige
sesión interactiva)
# Abrir manualmente:
echo "https://sso.marchernandez.es/api/authorize.php?\
client_id=mhsso_pki_client&\
redirect_uri=https://pki.marchernandez.es/api/auth/callback.php&\
response_type=code&\
scope=openid%20profile%20email&\
state=xyz123&\
code_challenge=$CHALLENGE&\
code_challenge_method=S256"

# Paso 3: tras login, el RP recibe ?code=AbCd123...
# Canjear code por tokens (esto sí desde el backend del RP):
curl -s -X POST https://sso.marchernandez.es/api/token.php \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=authorization_code" \
  -d "code=AbCd123..." \
  -d "client_id=mhsso_pki_client" \
  -d "client_secret=$CLIENT_SECRET" \
  -d "redirect_uri=https://pki.marchernandez.es/api/auth/callback.php" \
  -d "code_verifier=$VERIFIER" | jq .

# Respuesta:
# {
#   "access_token": "eyJhbGciOiJSUzI1NiIs...",
#   "token_type": "Bearer",
#   "expires_in": 3600,
#   "refresh_token": "ABab...",
#   "id_token": "eyJhbGciOiJSUzI1NiIs...",
#   "scope": "openid profile email"
# }
```

D.7.3 Consultar UserInfo con el `access_token`

```
curl -s https://sso.marchernandez.es/api/userinfo.php \
  -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIs..." | jq .

# Respuesta:
# {
#   "sub": "550e8400-e29b-41d4-a716-446655440000",
#   "username": "marc",
#   "email": "correo@marchernandez.es",
#   "display_name": "Marc Hernández Montesinos",
#   "role": "admin",
#   "totp_enabled": true,
#   "has_certificate": false,
#   "has_smartcard": false,
#   "last_login_at": "2026-05-12 13:08:42",
#   "created_at": "2026-02-06 17:43:18"
# }
```

D.7.4 Refresh del `access_token`

```
curl -s -X POST https://sso.marchernandez.es/api/token.php \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=refresh_token" \
  -d "refresh_token=ABab..." | jq .
```

D.7.5 Consultar OCSP y descargar CRL

```
# OCSP: comprobar el estado de un certificado emitido
openssl ocsp \
  -issuer ca/intermediate.crt \
  -cert mi-certificado.crt \
  -url http://ocsp.marchernandez.es \
  -resp_text

# (internamente: POST application/ocsp-request con la OCSPRequest DER en el
body)

# CRL: descargar la CRL más reciente de una CA específica
curl -s -o intermediate.crl http://crl.marchernandez.es/intermediate.crl

# Inspeccionar la CRL recién descargada
openssl crl -in intermediate.crl -inform DER -text -noout | head -40
```

D.8 Referencias

- Sección 7.5 *Bloque SSO* del cuerpo del TFG (flujo Auth Code + PKCE + OIDC).
- Sección 7.6 *Métodos de autenticación implementados* (mTLS, FNMT, DNIe).

- Sección 7.7 *Bloque OCSP* y Sección 7.8 *Bloque CRL* (justificación de rotación cada 4 h, modelo de respuesta).
- Sección 7.9 *Repositorio CA*.
- Sección 7.8 *Modelo de seguridad transversal* (PKCE, redirect_uri, rate limiting, CSRF, cabeceras).
- Anexo E (esquema de BD): tablas `sso_applications`, `sso_authorization_codes`, `sso_sessions`, `sso_keypairs`, `sso_audit_log`, `certificates`, `certificate_requests`, `crl_records`, `ocsp_queries`.
- Anexo F (perfiles OpenSSL): cómo `certificate_templates` se materializa en `.cnf`.
- RFC 6749 (OAuth 2.0), RFC 6750 (Bearer Tokens), RFC 7517 (JWK), RFC 7519 (JWT), RFC 7636 (PKCE), RFC 8252 (OAuth 2.0 for Native Apps), RFC 6960 (OCSP), RFC 5280 (X.509/CRL), OIDC Core 1.0, OIDC Discovery 1.0, OIDC RP-Initiated Logout 1.0.