KodeMed HIS REST Integration — Specification

Version: 0.3 (Aligned with OpenAPI — `CodingSessionRequest` / `HisWebhookPayload`) **Status:** Proposal — pending HIS partner validation **Date:** 2026-05-12 **Audience:** HIS partners integrating with KodeMed via REST + Webhook

---

1. Purpose

This document specifies a REST integration flow between KodeMed and a partner Hospital Information System (HIS). The flow is designed to be **generic** and supports multiple HIS partners through configuration only — no per-partner code branches.

2. Context

KodeMed currently offers three integration flows: DLL/COM, WebSocket/CodingClient, REST polling. None fits a web-based HIS partner that wants to:

- Start a coding session with clinical data via REST
- Open the KodeMed UI in an iframe or a new window
- Receive results via webhook when the coder confirms or discards
- Use their own Identity Provider (Keycloak) for authentication

This document defines a fourth flow that addresses these requirements.

3. Flow Architecture

4. Authentication

4.1 Identity Provider

The HIS partner provides a **Keycloak** IDP. KodeMed must be configured to accept JWT tokens issued by that Keycloak realm (specific issuer URI). KodeMed supports **multiple issuers in parallel** — every partner uses its own realm (planned for general use under **#491-D**; today every partner shares the existing KodeMed Keycloak).

4.2 Required token claims

Claim	Type	Required	Description
`sub`	string	yes	Unique user identifier
`email`	string	yes	User email (matched against KodeMed users)
`realm_access.roles[]` or `groups[]`	array	yes	Roles claim (claim path is configurable per IntegrationProfile: `oidcRolesPrimaryClaimPath` / `oidcRolesFallbackClaimPath`).
`exp`, `iat`, `iss`	standard	yes	Standard JWT claims

> **Status:** the multi-issuer wiring (`SecurityConfig` accepting tokens from each `IntegrationProfile.oidcIssuerUri`) is tracked under **#491-D**. Until that ships, the production server only validates tokens against its single configured issuer.

4.3 Roles — *Planned (#491-D)*

The role-to-capability mapping below is the **target contract**, tracked under **#491-D**. The current server does not enforce a `viewer` / `coder` split on the HIS REST entry point; any authenticated principal that the existing `CodingSessionController` accepts may open and complete sessions. Partners should provision realm roles in their IDP today so the switch is a config change later, not a re-deploy.

Role	Capabilities (target)
`coder`	Can open a session in editing mode (Apply enabled)
`viewer`	Can only view codes (Apply disabled)

4.4 `readOnly` flag precedence — *Planned (#491-D)*

A future `readOnly` flag on the create-session request will take priority over the token roles. The flag is **not yet** present on `CodingSessionRequest` ; today the only read-only path is `GET /api/v1/coding/session/last` (replays the user's last completed session in read-only mode).

`readOnly` flag (planned)	User role	Result
`false` (or omitted)	`coder`	Editing mode (Apply visible)
`false` (or omitted)	`viewer`	Read-only mode (Apply hidden)
`true`	`coder`	Read-only mode (Apply hidden) — flag wins
`true`	`viewer`	Read-only mode (Apply hidden)

5. API Endpoint Specification

5.1 Versioning

The API is versioned. Current version is `v1` . Non-backwards-compatible changes will produce a new version (`v2`).

Base path: `/api/v1/coding/` (HIS REST callers send `source=API` ; the same endpoint serves DLL and browser clients via different `source` values).

5.2 Create Session

Request:

```
POST /api/v1/coding/session HTTP/1.1
Host: server.kodemed.example
Authorization: Bearer <HIS-IDP-access-token>
Content-Type: application/json

{
  "data": "<spiges>...XML SPIGES content...</spiges>",
  "format": "spiges",
  "source": "API",
  "instanceId": "his-case-12345"
}
```

Field reference (see `CodingSessionRequest` in the OpenAPI):

Field	Type	Required	Description
`data`	string (XML)	yes	Case data in SPIGES format (max 5 MB)
`format`	string	yes for `source=API`	Wire format of `data`. Today the only supported value is `"spiges"`. DLL/UI callers may omit.
`source`	enum	yes for HIS REST	`"API"` for HIS REST integrations. DLL/UI clients send `"DLL"` / `"BROWSER"` (or omit; default is `DLL`).
`instanceId`	string (≤100 chars)	yes, **unique**	HIS-side correlator for the submitted `data` payload. One value per POST, regardless of how many SPIGES `<Fall>` cases ride inside `data` — they all belong to the same coding session and share this id. **MUST be unique across active sessions** — a second POST with an `instanceId` still in use returns `409 Conflict` (see §9.1). The id becomes free again once the original session completes / is cancelled / expires. Echoed back in the webhook as `instance_id` so the HIS matches the result against its original submission.

Webhook URL and HMAC secret are configured per-tenant on the KodeMed side via the IntegrationProfile admin API — they are not part of the create-session request.

Success response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "sessionId": "550e8400-e29b-41d4-a716-446655440000",
  "redirectUrl": "https://coding-ui.kodemed.example/spiges/session/550e8400-e29b-41d4-a716-44665544000
  "expiresAt": "2026-05-08T22:00:00Z"
}
```

Error responses:

HTTP	Reason	Body
400	Body invalid (missing field, malformed XML, `format` missing when `source=API`)	`{ "error": "VALIDATION_ERROR", "details": [...] }`
401	Token missing or expired	`{ "error": "UNAUTHENTICATED" }`
403	Token issuer not in allowlist, or insufficient role	`{ "error": "FORBIDDEN", "reason": "..." }`
409	(HIS REST) `instanceId` already in use by an active coding session	`CodingSessionResponse` with `hasConflict=true`, `conflictSessionId`, `conflictInstanceId`, `conflictMessage`
409	(DLL only) An active DLL session already exists for this user and `forceClose=false`	`CodingSessionResponse` with `hasConflict=true`
429	Rate limit exceeded	`{ "error": "RATE_LIMIT_EXCEEDED", "retryAfter": 60 }`
500	Internal error	`{ "error": "INTERNAL_ERROR", "traceId": "..." }`

5.3 Webhook callback (KodeMed → HIS)

Triggered when: Coder presses **Apply** or **Discard** on the session UI.

Request:

```
POST <IntegrationProfile.webhookUrl> HTTP/1.1
Authorization: HMAC-SHA256 t=1715205912,v1=<hex-sig>
Content-Type: application/json
Idempotency-Key: 550e8400-e29b-41d4-a716-446655440000:case.coded

{
  "event_type": "case.coded",
  "occurred_at": "2026-05-08T21:45:12Z",
  "session_id": "550e8400-e29b-41d4-a716-446655440000",
  "instance_id": "his-case-12345",
  "spiges": {
    "ent_id": "01.01.01.01",
    "burnr": "0000000001",
    "fall_id": "F-2026-00001"
  },
  "result_data": "<spiges>...modified XML SPIGES with codes...</spiges>"
}
```

> The `Authorization` header is present **only** when the IntegrationProfile is configured with `webhookSigning=HMAC_SHA256`. The signature is computed as `HMAC-SHA256(secret, "<t>.<body>")` and hex-encoded; the signed string is the unix timestamp `t`, a literal dot, and the raw request body. `Idempotency-Key` is always present (`<session_id>:<event_type>`) — use it for deduplication.

Field reference (see `HisWebhookPayload` in the OpenAPI):

Field	Type	Description
`event_type`	enum	`case.coded` (Apply) or `case.discarded` (Discard)
`occurred_at`	ISO-8601	When the coder completed the session
`session_id`	UUID v4	KodeMed session ID (echo of create response)
`instance_id`	string	Echo of original request `instanceId` — use this to match the result on the HIS side
`spiges`	object	SPIGES correlation triplet extracted from the result XML: `{ ent_id, burnr, fall_id }`
`result_data`	string (XML)	Modified SPIGES (non-empty when `event_type=case.coded`; omitted/empty when `event_type=case.discarded`)

Expected HIS response:

- `200 OK` → success, KodeMed marks delivery as completed
- `4xx` (except 408, 429) → permanent failure, no retry
- `5xx`, `408`, `429`, network error → KodeMed retries with exponential backoff (configurable, default 3 retries)

Idempotency: HIS should treat the webhook as **at-least-once delivery**. The `Idempotency-Key` header carries `<session_id>:<event_type>` and is stable across retries — HIS should deduplicate on it.

Signature verification (when HMAC enabled): The `Authorization: HMAC-SHA256 t=<unix>,v1=<hex-sig>` header carries the signature. To verify, the HIS recomputes `HMAC-SHA256(shared_secret, "<t>.<raw-body>")` (hex-encoded) and constant-time-compares to `v1`. Reject anything older than a small skew window (e.g. 5 min) to defeat replay. The HMAC secret is exchanged out-of-band when the IntegrationProfile is provisioned.

6. UI behaviour

6.1 Look & feel

The KodeMed UI shown via the create-session `redirectUrl` is the **same normal UI** as KodeMed. No "stripped-down" or "embedded" version — the user sees the full coding experience.

6.2 readOnly mode — *Planned (#491-D)*

The full readOnly behaviour below (request-flag-driven Apply suppression, READ ONLY badge wired to the HIS contract) is target behaviour under **#491-D**. Today the same UI is rendered for every HIS REST session; the readOnly badge only appears when the session is opened via `GET /api/v1/coding/session/last`.

When `readOnly: true` (planned):

- **Apply** button hidden
- All inputs disabled
- Visible **READ ONLY** badge in the header
- **Discard** (or "Close") button remains available to close the session

The `readOnly` flag is set by the HIS when the SPIGES cases have **already been coded** — the session is for review/audit only, not for editing.

6.3 Iframe vs new window

HIS can open the create-session `redirectUrl` either in:

- `<iframe>` (requires CSP `frame-ancestors` configuration for the HIS domain — per-tenant via `IntegrationProfile.embedCspFrameAncestors`, planned wiring under **#491-E**)
- New window/tab (`window.open`)

Authorized HIS domains for iframe must be configured on the KodeMed side per IntegrationProfile. Without this configuration, the iframe will be blocked by the browser.

7. Security

Aspect	Requirement
Transport	TLS 1.2+ mandatory for all endpoints
HIS access token storage	KodeMed never persists the inbound HIS access token; the webhook is signed by the per-tenant HMAC secret instead. The webhook secret itself is encrypted at rest via `EncryptedStringConverter`.
Webhook URL validation	HTTPS only; internal IP blocklist (RFC1918) to prevent SSRF
Webhook signature	`Authorization: HMAC-SHA256 t=<unix>,v1=<sig>` over `"<t>.<body>"`, with per-tenant secret. Only sent when `IntegrationProfile.webhookSigning=HMAC_SHA256`.
Rate limiting	Global token-bucket (`RateLimitProperties`), default **100 req/min** with stricter limits on auth endpoints. Per-IDP scoping is **planned (#491-D)**.
Audit log	Every HIS operation logged: request, response, webhook delivery, retries, errors
Token validation	Issuer in allowlist; signature vs JWK; `exp` not expired (multi-issuer allowlist planned under **#491-D**)

8. Required configuration

8.1 KodeMed needs from HIS

Item	Description	Example
Keycloak issuer URI	Full realm URL	`https://idp.his.example/realms/his-prod`
JWK Set URI	Auto-discovered from issuer	`.well-known/openid-configuration`
Webhook shared secret	For HMAC signature	(generated, exchanged out-of-band)
Allowed iframe origins	Domains from which HIS embeds the iframe	`https://app.his.example`
Expected token roles	`coder`, `viewer` (in `realm_access.roles[]` or `groups[]`)	—

8.2 HIS needs from KodeMed

Item	Description
KodeMed base URL	For the create-session call
KodeMed UI base URL	For the iframe (optional — embedded in `redirectUrl`)
Webhook secret	Same secret for signature verification

9. Error handling

9.1 Create-session errors

Scenario	KodeMed response	Recommended HIS action
Invalid SPIGES XML	`400` with `details`	Fix payload, retry
Missing `format` while `source=API`	`400` (`isHisRestFormatPresentWhenSourceIsApi` bean-validation message)	Add `format: "spiges"`, retry
Missing `instanceId`	`400` (`Instance ID is required`)	Add the HIS-side correlation identifier, retry
Token expired	`401`	Refresh token, retry
Token from wrong issuer	`403`	Check IDP configuration (multi-issuer support: #491-D)
Duplicate `instanceId` for an active session	`409 Conflict` with `hasConflict=true`, `conflictSessionId`, `conflictMessage` body	Use `conflictSessionId` to fetch the result of the original POST (`GET /api/v1/coding/session/{id}/result/data`), OR wait for it to complete and reuse the `instanceId` on a fresh submission. Never assume the second POST silently overwrites — it does not.
Active DLL session already open, `forceClose=false`	`200 OK` with `hasConflict=true` body	Inspect `conflictMessage` and either retry with `forceClose=true` or surface to the user
Rate limit exceeded	`429` with `retryAfter`	Wait and retry
KodeMed internal error	`500` with `traceId`	Contact KodeMed support with `traceId`

9.2 Webhook delivery errors

Scenario	KodeMed action
HIS returns 2xx	Mark delivered, no retry
HIS returns 4xx (except 408, 429)	Permanent failure, log, no retry
HIS returns 5xx	Retry with exponential backoff (default: 3 attempts at 5s, 30s, 120s)
HIS times out (> 30s default)	Retry as 5xx
HIS network error	Retry as 5xx
All retries exhausted	Persist to KodeMed dead-letter queue (`WebhookFailure`). Admin operations: `GET /api/v1/admin/webhook-failures`, `GET /api/v1/admin/webhook-failures/{id}`, `POST /api/v1/admin/webhook-failures/{id}/replay`.

10. Pre-implementation checklist (per HIS partner)

Validate the following with each HIS partner before scheduling implementation:

#	Item	Expected input
1	Keycloak realm URL (test + prod)	URI
2	Token claims structure: `realm_access.roles` or `groups`	enum
3	User mapping: identify coder by `sub` or `email`	enum
4	HMAC signature support on the HIS webhook endpoint	yes / no / planned
5	Rendering: iframe or new window (drives CSP configuration)	enum
6	SPIGES version supported by HIS	semver
7	Test environment available for end-to-end integration	yes / no
8	Expected volumes (sessions/day, hourly peaks, working hours)	numbers
9	Required SLA on API availability	best-effort / 99.x %
10	Case status mapping on HIS side (equivalent of "in coding" / "coded" / "discarded")	enum

11. Roadmap & milestones

Implementation is planned in a dedicated milestone. Phases:

1. **Specification approval** (this document) — joint review with HIS partner
2. **Backend implementation** — new endpoint, multi-issuer OIDC, webhook trigger, audit log
3. **Frontend implementation** — readOnly mode, iframe support, badge UI
4. **Integration testing** — joint test with HIS test environment
5. **Documentation finalisation** — API docs (OpenAPI), integration guide, deployment guide
6. **Production rollout** — phased: pilot hospital → general availability

12. Future extensibility

The flow is designed as **generic** to support multiple HIS partners. Each partner is added through configuration only:

- Entry in `KODEMED_HIS_ALLOWED_ISSUERS` (their Keycloak realm)
- Entry in `KODEMED_HIS_ALLOWED_ORIGINS` (their domain, for iframe embedding)
- Their own shared secret for HMAC

No code changes are required to onboard a new HIS partner.

13. Glossary

Term	Definition
HIS	Hospital Information System — partner clinical software
IDP	Identity Provider (here: Keycloak)
SPIGES	Swiss Patient & Inpatient Grouper for Electronic Submission — XML format for clinical case data
Coder	Medical coder — user who assigns ICD-10 / CHOP codes
Webhook	HTTP callback from KodeMed to HIS when a session completes
JWT	JSON Web Token — bearer token format used for OIDC
HMAC	Hash-based Message Authentication Code — used for webhook signature
CSP	Content Security Policy — browser security header controlling iframe embedding

14. Document control

Version	Date	Author	Change
0.1	2026-04-25	Ricardo Mieres / KodeMed	Initial draft
0.2	2026-05-08	Ricardo Mieres / KodeMed	Generic, partner-agnostic version
0.3	2026-05-12	Ricardo Mieres / KodeMed	Align with OpenAPI: endpoint is `/api/v1/coding/session`; request fields renamed (`caseData` → `data`, add `format` / `source`, `correlationId` → `instanceId`); webhook payload renamed to `HisWebhookPayload` shape (`event_type`, `occurred_at`, `session_id`, `instance_id`, `spiges{ent_id,burnr,fall_id}`, `result_data`); webhook URL + HMAC moved out of the request into the per-tenant IntegrationProfile (#491, #493). HMAC header format corrected to `Authorization: HMAC-SHA256 t=<unix>,v1=<sig>` signed over `"<t>.<body>"`; idempotency carried by `Idempotency-Key: <session_id>:<event_type>` (was non-existent `X-KodeMed-Signature` / `X-KodeMed-Delivery`). Response field corrected `sessionUrl` → `redirectUrl`. §4.3 (viewer/coder roles), §4.4 (readOnly precedence), §6.2 (readOnly UI), §6.3 (iframe CSP per-tenant), §7 (per-IDP rate limit) explicitly marked **Planned (#491-D / #491-E)** since the server does not enforce them today.

15. Contact

KodeMed: Ricardo Mieres — `ricardo@mieresit.com` **HIS partner:** to be filled by partner