KodeMed

Systemarchitektur

Für CTOs, IT-Administration & DevOps

Version 2026.5.15.59502 | KodeMed AG

Systemübersicht

KodeMed-Plattformkomponenten

KodeMed.Server

Java-Backend (REST + WebSocket)

KodeMed.DataServer

Klassifikationsdaten-API

KodeMed.GrouperServer

DRG-Gruppierungs-API (SwissDRG, TARPSY, ST Reha)

KodeMed.CodingUI

React/Vite-Web-Frontend

KodeMed.Interface

COM-DLL (.NET 9.0)

KodeMed.CodingClient

System-Tray-App – portable EXE
WebSocket, Webhook, Sprachauswahl

```
Ablauf: KIS/Drittanbieter → Server (REST/WS) → CodingClient (WS) → DLL → CodingUI (WebView2)

Ablauf: KIS (direkt) → DLL (COM) → Server → CodingUI (WebView2)
```

KIS-Integration

Integration des Klinikinformationssystems via COM-DLL

DLL-API (IKodeMed)

Method	Description
SendConfig(xml)	Load configuration
DoCodingWithFormat(data, format)	Full UI flow (auth + browser)
DoCoding(data, format)	Headless processing
GetResults()	Retrieve coding results
GetResultData()	Get clean SpiGes XML
DisposeClient()	Release resources

Integrationsbeispiel (C#)

```
// 1. Create instance
var coding = new Coding();

// 2. Configure
coding.SendConfig(configXml);

// 3. Process with UI
coding.DoCodingWithFormat(
  spigesData, FormatType.SpiGes);

// 4. Check action
if (coding.LastCodingAction
    == CodingAction.Applied)
{
  var results = coding.GetResults();
  var xml = coding.GetResultData();
}

// 5. Cleanup
coding.DisposeClient();
```

Client-Installation

Drei Bereitstellungsmethoden

MSI-Installer (empfohlen)

- KodeMed.msi-Paket
- Pro Benutzer (kein Admin) oder pro Maschine (Citrix)
- Stille Installation via GPO/SCCM/Intune
- COM-Registrierung + Konfiguration enthalten
- WebView2-Runtime erforderlich

Manuell / Xcopy

- DLL + Abhängigkeiten kopieren
- Manuelle COM-Registrierung
- Umgebungsvariablen manuell setzen
- Für erfahrene Benutzer

CodingClient

- Installiert via MSI
- System Tray + WebSocket
- Auto-Reconnect + Webhook
- Citrix-/VDI-kompatibel
- WebView2-basierte Oberfläche

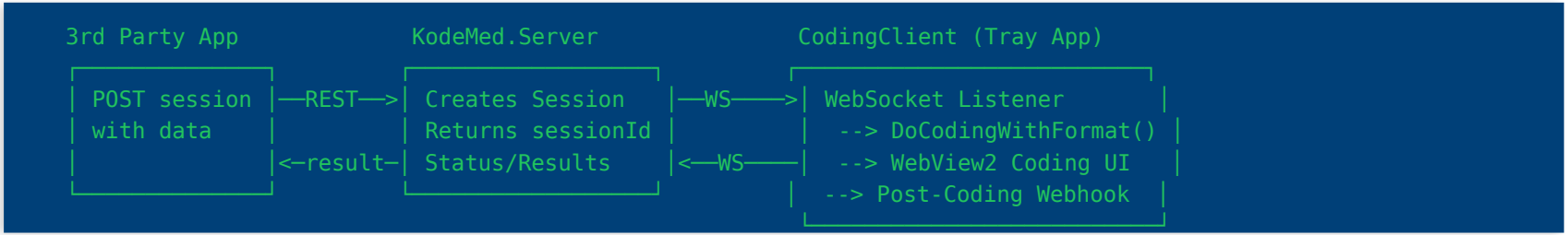
Konfiguration (automatisch gesetzt durch MSI-Installer oder beim ersten Start)

Variable	Description	Example
KODEMED_HOME	Installation directory	C:\Users\joe\KodeMed\
KODEMED_DLL_PATH	Full path to KodeMed.dll	C:\Users\joe\KodeMed\KodeMed.dll
KODEMED_EXE_PATH	Path to CodingClient.exe	C:\Users\joe\KodeMed\KodeMed.CodingClient.exe

CodingClient-Architektur

System-Tray-Anwendung für Drittanbieter-Integration

Integrationsablauf



Funktionen

- Einzelinstanz (globaler Mutex)
- WebSocket-Listener für CODING_SESSION_LAUNCH
- Post-Coding-Webhook (Fire-and-Forget, Retry)
- Sprachauswahl (DE/FR/IT/EN, Tray-Menü)
- OAuth2-Anmeldung/-Abmeldung via Tray-Menü
- Autostart bei Windows-Anmeldung (HKCU\Run)
- Sitzungen aus lokalen Dateien öffnen

Citrix-/VDI-Highlights

- Eigenständig: keine .NET-Runtime-Abhängigkeit
- Alle Schreibvorgänge auf Benutzerebene (HKCU, Umgebungsvariablen)
- Portabel: Ordner kopieren und starten
- COM via HKCU\Software\Classes (kein Admin)
- Erneutes Setup bei nicht-persistenter VDI-Anmeldung

Serverarchitektur

Java-Backend mit REST-API und WebSocket

KodeMed.Server (Java)

- Spring-Boot-Anwendung
- REST-API: Konfiguration, Sitzungen, Audit, Instanzen, Layouts, Health
- WebSocket-Endpunkt (/ws/dll, /ws/app)
- JPA/Hibernate-Persistenz (H2/PostgreSQL)
- Audit-Trail (Sitzungsereignisse, Benutzeraktionen)
- Layout-Verwaltung (Speichern/Laden/Teilen)
- OAuth2-Token-Validierung
- Grouper-Integration (SwissDRG)
- Docker-Bereitstellung

KodeMed.DataServer

- Klassifikationsdaten-API
- ICD-10-GM-Codes + Beschreibungen
- CHOP-Prozedurcodes
- ATC-Medikamentencodes
- SwissDRG-Grouper-Katalog
- Volltextsuche mit Autovervollständigung
- Versionierte Kataloge (jährliche Updates)
- REST-API mit Caching

Kodierungs-UI

React/Vite-Web-Frontend

KodeMed.CodingUI (React + Vite)

- React 18 mit TypeScript + Vite
- Zustand State Management
- 14+ Datenblöcke (verschiebbar, react-grid-layout)
- Bearbeitbare Code-Tabellen (ICD, CHOP, ATC)
- Code-Suche mit Autovervollständigung
- SpiGes-Attributbearbeitung pro Zeile
- Administrative Datenformulare
- Echtzeit-Grouper-Ergebnisse (SwissDRG)
- Mehrsprachige UI (DE/FR/IT/EN)
- Undo-Verlaufsleiste (visueller Schritt-durch)
- Eingebettet in DLL via WebView2
- Standalone-Browsermodus (OAuth2/OIDC)
- WebSocket-Live-Updates
- Eigene Layouts speichern/laden/teilen
- Responsives Design + Dark/Light Theme
- Tastaturnavigation
- Export-/Import-Funktionalität

Authentifizierung & Sicherheit

OAuth2 / OpenID Connect

OAuth2 / OIDC

- Authorization Code + PKCE-Verfahren
- Keycloak, Auth0, Azure AD kompatibel
- DLL: eingebetteter Browser-Login
- UI: Standard-OIDC-Redirect
- Token-Auto-Refresh
- Mandantenfähig via Realms
- Gruppenbasierte Zugriffskontrolle
- OpenID-Connect-Benutzerclaims

Sicherheitsmassnahmen

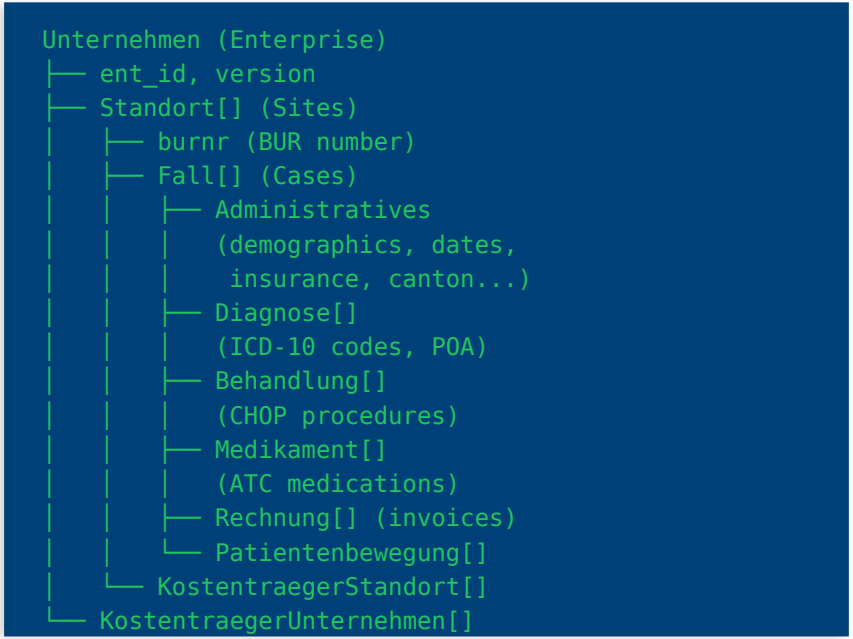
- HTTPS überall (TLS 1.2+)
- WSS für WebSocket-Verbindungen
- JWT-Token-Validierung auf dem Server
- CORS-Policy-Durchsetzung
- Sitzungsablauf (konfigurierbar)
- Audit-Logging (alle Sitzungsereignisse)

DSGVO-/nDSG-Konformität

- Datenminimierung als Standard (Webhook Opt-in)
- includeResultData/includeGrouperResults: aus
- SSRF-Schutz bei Webhook-URLs
- Keine Geheimnisse in Client-Konfiguration

Datenmodell

SpiGes-XML-Struktur (BFS / Bundesamt für Statistik)



Wichtige Entitäten

Entity	Key Fields
Administratives	abc_fall, alter, geschlecht, ein/austritt
Diagnose	diagnose_kode (ICD-10), diagnose_poa
Behandlung	behandlung_chop (CHOP), behandlung_beginn
Medikament	medi_atc, medi_dosis, medi_einheit
GrouperResult	DRG, MDC, PCCL, cost weight

Post-Coding-Webhook

Fire-and-Forget-HTTP-Callback nach Kodierungssitzungen

Funktionsweise

- Kodierer schliesst Sitzung ab (Anwenden/Verwerfen/...)
- CodingClient sendet JSON per POST an konfigurierte URL
- Asynchron: blockiert den Kodierer nicht
- Retry mit exponentiellem Backoff (1s→2s→4s...60s)
- Kein Retry bei 4xx (ausser 429)

Konfiguration (Umgebungsvariablen)

Variable	Default
KODEMED_HOOK_ENABLED	false
KODEMED_HOOK_URL	(empty)
KODEMED_HOOK_INCLUDE_RESULT_DATA	false
KODEMED_HOOK_INCLUDE_GROUPER_RESULTS	false
KODEMED_HOOK_EVENTS	applied

Webhook-Payload

```
POST https://his.example.com/api/results
Content-Type: application/json

{
  "eventType": "coding_session_completed",
  "sessionId": "sess_abc123",
  "codingAction": "Applied",
  "applied": true,
  "username": "dr.muller",
  "sourceFormat": "SpiGes",
  "casesProcessed": 3,
  "diagnosesCount": 12,
  "treatmentsCount": 7,
  "resultData": "...(opt-in)...",
  "grouperResults": [...(opt-in)...]
}
```

Auth: Bearer-Token oder Custom-Header (nur lokale Konfiguration)

Bereitstellung

Docker, On-Premise, Cloud

Serverkomponenten

```
# Docker Compose
services:
  kodemed-server:
    image: kodemed/server:latest
    ports: ["8080:8080"]

  kodemed-dataserver:
    image: kodemed/dataserver:latest
    ports: ["8081:8081"]

  kodemed-grouper-server:
    image: kodemed/grouper-server:latest
    ports: ["8082:8082"]

  kodemed-ui:
    image: kodemed/codingui:latest
    ports: ["3000:3000"]

# OIDC Provider (external IdP, not bundled)
# https://customer-sso.example.com/auth
```

Client-Bereitstellung

- CodingClient: portable, eigenständige EXE
- In beliebigen Ordner kopieren und starten
- Automatische DLL- + Umgebungsvariablen-Registrierung
- Citrix/VDI: als App oder im Image veröffentlichen
- Gruppenrichtlinie: KODEMED_HOME-Umgebungsvariable setzen
- Stilles Ersteinrichtungs-Setup
- Konfigurationsüberschreibung: kodemed-client-config.json
- Auto-Update: für zukünftige Versionen geplant

Bereitstellungs-Topologie

Kubernetes / OpenShift mit Harbor Registry

Harbor Registry

- kodemed/server:latest
- kodemed/dataserver:latest
- kodemed/grouper:latest
- kodemed/codingui:latest

Trivy scan + Cosign/Notary

pull >>>

Kubernetes / OpenShift

Ingress / Route

Service	HPA	Port
kodemed-server	Yes	:8080
kodemed-dataserver	Yes	:8081
kodemed-grouper	-	:8082
kodemed-ui	-	:3000

PostgreSQL (StatefulSet / external)

- Helm Charts oder einfache K8s-Manifests
- Horizontales Pod-Autoscaling (HPA) für Server & DataServer

- Private Registry für Image-Speicherung & -Verteilung
- Schwachstellenanalyse (Trivy) bei jedem Push

Konfiguration

XML-/JSON-Konfiguration, Umgebungsvariablen, Webhook

DLL-/Client-Konfiguration (XML oder JSON)

```
<KodeMedConfig>
  <ServerUrl>https://api.example.com
  </ServerUrl>
  <CodingUIUrl>https://ui.example.com
  </CodingUIUrl>

  <!-- OAuth2 -->
  <OAuth2Url>https://sso/auth</OAuth2Url>
  <OAuth2Realm>kodemed</OAuth2Realm>
  <OAuth2ClientId>kodemed-dll</OAuth2ClientId>

  <!-- Language (de/fr/it/en, auto-detect) -->
  <Language>de</Language>

  <!-- Processing -->
  <OutputFormat>XML</OutputFormat>
  <ValidateSchema>true</ValidateSchema>
</KodeMedConfig>
```

Wichtige Einstellungen

Setting	Default	Description
ServerUrl	-	Main server URL
OAuth2Url	-	Identity provider
Language	auto	de, fr, it, en
OutputFormat	XML	XML or JSON
SessionExpiry	60	Minutes
OfflineTimeout	5	Minutes

Webhook-Umgebungsvariablen (Server)

Variable	Default
KODEMED_HOOK_ENABLED	false
KODEMED_HOOK_URL	(empty)
KODEMED_HOOK_EVENTS	applied
KODEMED_HOOK_RETRY_COUNT	3

Schnellstart

In 3 Schritten betriebsbereit

Schritt 1: Server bereitstellen

```
docker compose up -d    # Starts Server, DataServer, GrouperServer, CodingUI
```

Schritt 2: Client installieren (eine Option wählen)

```
# Option A: CodingClient (recommended)
# Just run KodeMed.CodingClient.exe
# Auto-setup handles everything
```

```
# Option B: MSI silent install (GPO/SCCM)
msiexec /i KodeMed.msi /quiet SERVERURL="https://..."
```

Schritt 3: Kodierung starten

```
var coding = new Coding();
coding.SendConfig(File.ReadAllText("kodemed-config.xml"));
coding.DoCodingWithFormat(spigesXml, FormatType.SpiGes);     // Opens auth + coding UI
string results = coding.GetResults();                         // XML or JSON results
```