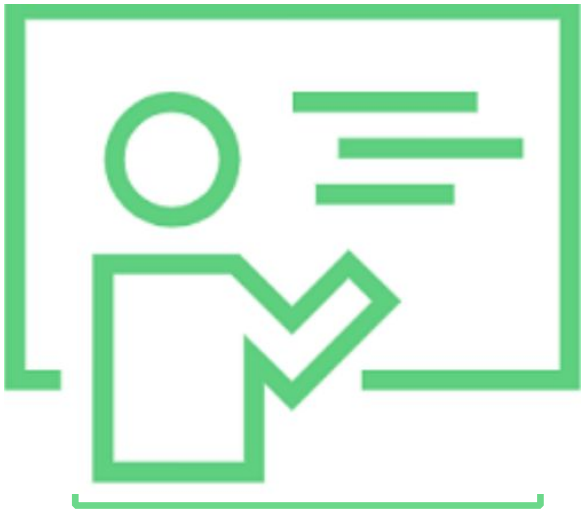

JSON, XML och datakontrakt

Joakim Englund, Systementor AB

Repetition från dag 3

Vad bör ett API-kontrakt beskriva?

- HTTP-metod och URL
- Headers och `Content-Type`
- Request och response
- Statuskoder och felsvar
- Datans struktur

Samma mätning, flera format

Mätdata → JSON
→ XML

Dagens mål

- Känna igen JSON- och XML-struktur
- Serialisera och parsa sensordata
- Validera fält, typer och rimlighet
- Formulera ett tydligt datakontrakt
- Felhantering

Datamodell och representation

Intern datamodell (typ en struct eller klass där vi lagrar datat)

Mätdata { sensorId, value, unit }
↓ serialisering ↑ parsning
JSON eller XML

JSON-struktur

```
{
  "sensorId": "temp-01",
  "value": 21.7,
  "unit": "C"
}
```

- Objekt med namn–värde-par
- Tal och strängar är olika typer
- Arrayer samlar flera värden

Mer komplext exempel:

```
{
  “metadata”: {
    "sensorId": "temp-01",
    "unit": "C"
  },
  “actual-value”: {
    “value”: 21.7
  }
}
```

XML-struktur

```
<?xml version=”1.0” encoding=”UTF-8”?>
<reading>
  <sensorId>temp-01</sensorId>
  <value>21.7</value>
  <unit>C</unit>
</reading>
```

- Ett rotelement
- Nästlade element och attribut
- Textvärden måste tolkas

Mer komplext exempel:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<reading>
  <metadata>
    <sensorId>temp-01</sensorId>
    <unit>C</unit>
  </metadata>
  <actual-value>
    <value>21.7</value>
  </actual-value>
</reading>
```

JSON vs XML, repetition

JSON	***XML***
Kompakt	Mer explicit struktur
Inbyggda grundtyper	Text måste ofta typas av kontraktet
Vanligt i REST-API:er	Centralt i SOAP och många äldre typer av integrationer
JSON Schema	XSD

Tre nivåer av validering

- Syntax - Går texten att parsa?
- Struktur och typ - Finns obligatoriska fält med rätt typ?
- Semantik och rimlighet - Är värdet meningsfullt i domänen?

Datakontraktet

Fält	***Typ***	***Obligatoriskt***	***Regel***
sensorId	sträng	ja	får inte vara tomt
value	tal	ja	-50 till 100
unit	sträng	ja	`C` för temp i Celsius

Begripliga fel

```
{
  "error": "value must be a number"
}
```

Ett bra fel svarar på:

- Vad var fel?
- Var fanns felet?
- Hur kan avsändaren korrigera det?

Content-Type väljer tolkning

application/json → JSON-parser
application/xml → XML-parser
↓
samma Mätdata i slutändan

Schema

- JSON Schema beskriver regler för JSON
- XSD beskriver struktur och typer för XML
- Ett schema kan användas för att automatisera delar av valideringen
- Domänregler kan fortfarande kräva programlogik

JSON Schema

JSON Schema beskriver vilka fält och datatyper ett JSON-dokument ska innehålla

Ex:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "sensorId": { "type": "string" },
    "value": { "type": "number", "minimum": -50, "maximum": 100 },
    "unit": { "type": "string", "enum": ["C"] }
  },
  "required": ["sensorId", "value", "unit"]
}
```

XSD

Står för XML Schema Definition - precis vad det låter som

Ex:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="reading">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="sensorId" type="xs:string"/>

      <xs:element name="value">
        <xs:simpleType>
          <xs:restriction base="xs:decimal">
            <xs:minInclusive value="-50"/>
            <xs:maxInclusive value="100"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="unit" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Datakontraktet i IoT-kedjan

Sensor → validering → nätverk → validering → lagring → beslut/aktuator

- Små payloads sparar överföring och minne
- Enhetsidentitet och enhet måste vara entydiga
- Feldata ska stoppas innan styrning
- Gatewayen kan översätta JSON till XML

Dagens laboration

1. Lås datakontraktet
2. Skapa giltig JSON och XML
3. Parsa till vår interna datamodell
4. Validera fält, typer och rimlighet
5. Testa och dokumentera fel
6. Koppla till API:t

Demo: JSON och XML

Förevisar serialisering och parsning