# Introduction

The CPSV-AP API implemented for Netherlands converts the output of SRU API into CPSV-AP data model.

In order to perform such conversion a mapping analysis was done to understand which data could be extracted from the SRU API (insert real link of the api and check in the code).

After that, the CSPV-AP API have been implemented starting for the analysis of the input request and the output response, which determined the API contract.

## The CPSV-AP Mapping

The CPSV-AP mapping with the SRU API is based on multiple XML schemas mainly:

* Owms.xsd (5 properties can be mapped to CSPV-AP)
* Sc.xsd (2 properties can be mapped to CSPV-AP)
* Gzd.xsd (2 properties can be mapped to CPSV-AP)

The outcome of the mapping analysis shows that, except for 2 properties (authority and productHTML) which cardinalities are more relaxed than CPSV-AP, a mapping towards CPSV-AP is possible, however the result might not be fully compliant with CPSV-AP.

The reader can find more details in the attached spreadsheet.

## API Architecture

### Input request and Output response

The CSPV-AP API implemented is a REST API which has 3 input parameters:

* startRecord
* MaximumRecord
* Query

Which are the same input for the SRU API and 1 output:

* PublicServiceDataset

INSERT HERE A SEQUENCE DIAGRAM BETWEEN CPSV-API and Netherlands API.

The PublicServiceDataset will contain a list of PublicService which in turn will have the following properties/relations (based on the aforementioned CSPV-AP mapping):

* Identifier
* Title
* Language
* Spatial
* hasCompetentAuthority
* type
* description

While the PublicOrganization connected will have its identifier and the Spatial property.

INSERT HERE AN EXAMPLE OF OUTPUT

The JSON-LD output take advantage of the CPSV-AP JSON-LD context to determine the linked data relations.

INSERT HERE A SWAGGER SCREENSHOT OF THE CONTRACT

Once the contract was determined, the API structure has been generated via a Maven Swagger codegen.

## Formats

The CPSV-API support content negotiation in order to provide different output formats via the Accept header by using the related mime types:

|  |  |
| --- | --- |
| Format | Mime type |
| XML | application/xml |
| JSON |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Check if the API provides JSON-LD as default output.

## Component architecture

INSERT HERE A COMPONENT ARCHITECTURE

Describe:

1. Applicationcontext.xml
2. Binary Providers (GenericReport)
3. SRU Object – Response
4. JSON-LD Annotations