OpenBudgets.eu

OpenBudgets.eu: Fighting Corruption with Fiscal Transparency

Project Number: 645833 Start Date of Project: 01.05.2015 Duration: 30 months

# Deliverable 4.1

## Specification of services' Interfaces

| | |
|---|---|
| Dissemination Level | Public |
| Due Date of Deliverable | Month 4, 30.08.2015 |
| Actual Submission Date | 04.09.2015 |
| Work Package | WP 4, OpenBudgets.eu Requirements, Platform Architecture Integration and Development |
| Task | T 4.1, Specification of services' Interfaces |
| Type | Report |
| Approval Status | Final |
| Version | 1.0 |
| Number of Pages | 16 |
| Filename | D4.1 Specification of services' Interfaces |

**Abstract:** A detailed report defining the specifications for the interfaces for WP2, WP3 and WP7, based on D4.2. – Analysis of the required functionality



Project funded by the European Union's Horizon 2020 Research and Innovation Programme (2014 – 2020)

## History

| Version | Date | Reason | Revised by |
|---|---|---|---|
| 0.1 | 17.08.2015 | First Draft for Documentation | Fahrettin Gökgöz |
| 0.2 | 29.08.2015 | Paul Walsh's Review | Paul Walsh |
| 0.3 | 01.09.2015 | Internal Review | Fabrizio Orlandi |
| 1.0 | 03.09.2015 | Final Deliverable | Sören Auer |

## Author List

| Organisation | Name | Contact Information |
|---|---|---|
| UBONN | Fahrettin Gökgöz | gokgozf@gmail.com |
| Fraunhofer | Fabrizio Orlandi | fabrizio.orlandi@iais.fraunhofer.de |
| Fraunhofer | Sören Auer | soeren.auer@iais.fraunhofer.de |
| OKFN | Paul Walsh | Paul.walsh@okfn.org |

# Executive Summary

The purpose of this document about services' interfaces is to define the general software architecture of the OpenBudgets.eu platform. In addition to the architecture described in Deliverable 4.2 "Analysis of the Required Functionality of OpenBudgets.eu", more detailed information about each module in the architecture is defined. The primary objective is to describe how the modules are interacting between each other.

Each module is responsible for a specific task, either related to a technical or a financial perspective. For instance, Data Load, Write API, and DataStore modules are mostly related to technical tasks. At the same time, analytics and visualizations are largely related to a financial level of analysis and manipulation of the data. Section 2 "Interfaces" provides the necessary information about each interface exposed by the OpenBudgets.eu software components.

# Abbreviations and Acronyms

| | |
|---|---|
| **API** | Application Program Interface |
| **CSV** | Comma Separated Value |
| **JSON** | JavaScript Object Notation |
| **LOD** | Linked Open Data |
| **OLAP** | Online Analytical Processing |

**Table 1: Acronyms**

# Table of Contents

# List of Figures



# List of Tables

# 1 Introduction

OpenBudgets.eu aims at defining an open government data analysis and presentation framework. This section summarizes the general architecture for the OpenBudgets.eu framework by introducing the software modules required in this project. Their specific interfaces will be introduced in the following section (Section 2). The aim of this section is to provide the reader an overview of the framework.

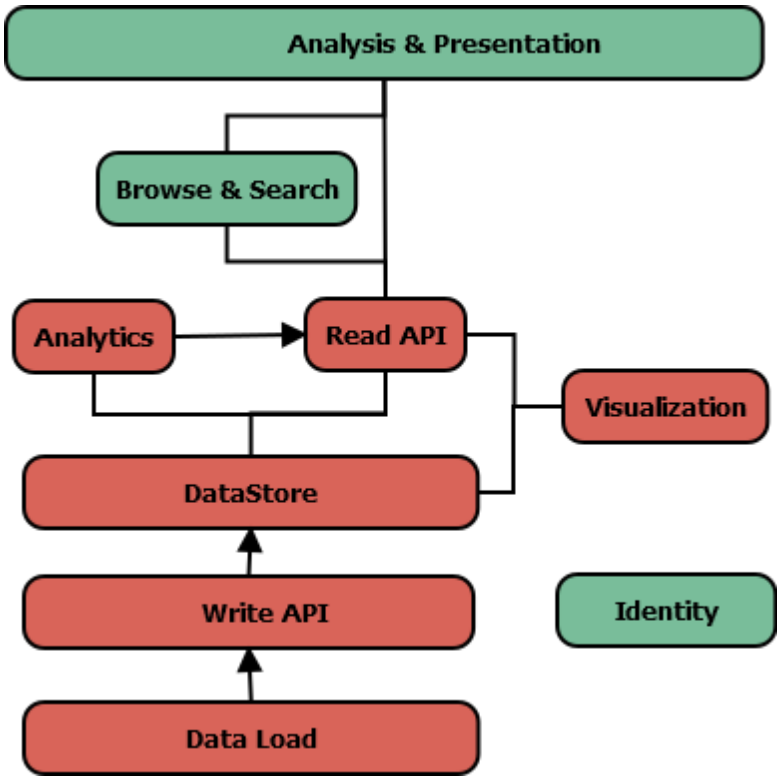


**Figure 1 Architecture and Interfaces Overview**

OpenBudgets.eu (as shown in Figure 1) includes the following modules (from bottom to top): Data Load, Write API, DataStore, Analytics, Read API, Visualization, and Browse & Search. Each module fulfils a specific and self-contained purpose, this is to allow for a modular software architecture. In the following section we will detail the aforementioned modules in the same order, starting from the Data Load one. In addition to a short description of the modules we include the definition of the software interfaces exposed to the other connected components. For this deliverable Specification we followed the following reference: "Documenting Software Architectures: Organization of Documentation Package" and the OpenSpending reference architecture[1].

[1] http://community.openspending.org/next/

# 2 Interfaces

## 2.1 Data Load Module

The Data Load module is responsible for the tasks of data acquisition, cleaning, and loading into the Data Store of the OpenBudgets.eu platform. It is directly connected to the Write API as data flows to the Data Store in “write” mode through the methods defined in this API.

### 2.1.1 Interface Identity

The Data Load module consists of three identity interfaces: one for adding data files to the Data Store (“AddFile”), one for modelling the data (“ModelData”), and one for publishing the inserted data (“Publish”), as depicted in Figure 2.

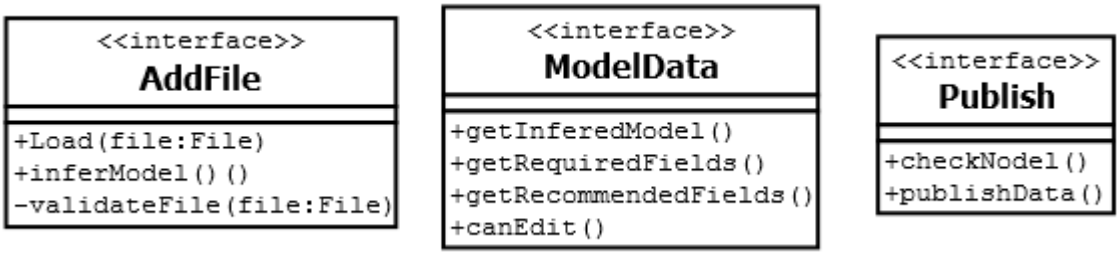


**Figure 2: Interfaces of the Data Load module**

### 2.1.2 Resources Provided

The Data Load module provides access to the loaded and validated data files. These files can be modified according to needs, and also loaded directly into the platform’s pipeline.

### 2.1.3 Locally Defined Data Types

The most important data types in this module are those included in the Data Cube Vocabulary[2] for financial data, and the OpenSpending Data Package format for describing datasets and metadata about them.

[2] http://www.w3.org/TR/vocab-data-cube/

### 2.1.4 Error Handling

The Data Load module has basic error handling capabilities. In particular about the initial validation of the loaded data file. Additionally, the module checks the mappings for the given fields with the loaded file.

### 2.1.5 Variability Provided

The Data Load module allows the possibility to load arbitrary budget data into the system. The inserted data has to comply only to a few specific given fields, as described in this section.

### 2.1.6 Quality Attribute Characteristics

Basic file quality checking is performed when importing the data. For example, whether a numerical “amount field” actually contains numerical values or not. The data import procedure is not completed if data does not comply with the requirements.

### 2.1.7 What is the Element Requirement

The Data Load module requirements can be grouped into the following categories: at the most basic level (tabular data format) a CSV file is required with a predefined schema; at a higher Linked Data (or graph-based data format) level, a pipeline for RDF to/from CSV transformations is required.

### 2.1.8 Rationale and Design Issues

The idea behind the Data Load module is that the files will be checked for simple validation and then will be stored in the data storage. This way, additional validation extensions can be reduced (or are not necessary) for the rest of the entire pipeline.

### 2.1.9 Usage Guide

Each publicly-available budget data which is inserted into the platform has to be processed by the Data Load module. Therefore the module allows users to create and load data packages from CSV, or TTLs with a pre-processing pipeline. During these operations, the module also provides basic validity checks, and reducing the required knowledge about the budget data types.

## 2.2 DataStore Module

The DataStore module is responsible for storage and persistence of the OpenBudgets.eu platform data. It is one of the core modules of the architecture and centrally connected to many of the other modules: Write and Read APIs, Visualization and Analytics.

### 2.2.1 Interface Identity

The DataStore module has four interface identities: one for the input of data, in order to load CSV and JSON files from the Data Load module into the flat files storage system; one for the raw output, in order to directly stream files from the flat files storage system; one for passing pre-calculated flat file data for simple visualization; and one for Analytics/Read API to calculate rich queries over the derived database, e.g. an OLAP store.

### 2.2.2 Resources Provided

The DataStore module provides flat files as main resources. Each of the modified backend storage points is a variation of these provided files.

### 2.2.3 Locally Defined Data Types

The most important data types in this module are those included in the following vocabularies or schemas: Data Cube Vocabulary, Json Table Schema, OLAP Cubes, OpenSpending Data Package.

### 2.2.4 Error Handling

Error handling is not defined for this module.

### 2.2.5 Variability Provided

The DataStore module stores the data in given flat-files. The provided OLAP, SQL, and other alternative backend options are variables for this storage module.

### 2.2.6 Quality Attribute Characteristics

The DataStore module officially does not define any quality attribute about performance for reading and accessing the data. Quality of the data is guaranteed by the underlying storage technical layer and the Data Load & Write API modules.

### 2.2.7 What is the Element Requirement

The DataStore module requires cleaned and validated data to be stored and further processed.

### 2.2.8 Rationale and Design Issues

The idea behind the DataStore module is to provide a common access point for the data, and to offer easy-to-modify alternatives for different types of backend storages. The flat files system will be the basis of the storage system and on top of it different, more complex alternative storage systems (e.g. triplestores) could be implemented.

### 2.2.9 Usage Guide

The DataStore module is a fully technical module used for storing both (i) the validated budget data, and (ii) previously calculated analytics. Each data access operation performed from the Read API, or the Visualization module, invokes the derived data access components of DataStore to collect the data.

## 2.3 Write API

Write API is an intermediator between Data Load and DataStore modules.

### 2.3.1 Interface Identity

Write API does not specify any interface other than the ones defined in Data Load and DataStore modules.

### 2.3.2 Resources Provided

Write API does not provide a resource but rather use the defined ones.

### 2.3.3 Locally Defined Data Types

Locally defined data type does not exist.

### 2.3.4 Error Handling

Error handling is not defined for this module.

### 2.3.5 Variability Provided

Variability is not defined for this module.

### 2.3.6 Quality Attribute Characteristics

Quality of the module can be measured based on the defined authorization and authentication methods.

### 2.3.7 What is the Element Requirement

Module requires the data previously validated and loaded by the Data Load module.

### 2.3.8 Rationale and Design Issues

The idea behind the module is to create a control mechanism for the user.

### 2.3.9 Usage Guide

One obvious usage is the separation of the usage of local user data, and also the access control for the stored data.

## 2.4 Read API & Analytics Modules

Read API and Analytics are two conceptually separated modules in OpenBudgets.eu architecture. However, here they are combined and represented as with the same implementation. They actually share the same features and configuration from the interfaces perspective. The Read API is responsible for each individual read operation on the system, and the Analytics module is responsible for creating aggregations using arbitrary queries on the DataStore.

### 2.4.1 Interface Identity

The Read API module requires a data transferring interface for the DataStore module and the Analytics module. It also provides a common data access interface for OLAP cubes and other types of advanced queries.

### 2.4.2 Resources Provided

Through the Read API and Analytics modules enriched data access is made available for OLAP cubes.

### 2.4.3 Locally Defined Data Types

In these modules, locally defined data types are not specified.

### 2.4.4 Error Handling

Error handling is not defined for these modules

### 2.4.5 Variability Provided

The Read API and Analytics modules define concrete and controlled access operations. Therefore variations are not provided in the same version. The reason behind this choice is to keep the design stable for each version.

### 2.4.6 Quality Attribute Characteristics

Quality of the Read API and Analytics module can be measured based on the expressive power for the implemented functionalities.

### 2.4.7 What is the Element Requirement

In order for data to be accessible via the Analytics/Read API, it first needs to be loaded into the DataStore. Hence, the required element for these modules is the data endpoint of the DataStore where to run the queries.

### 2.4.8 Rationale and Design Issues

Read API provides unified OpenBudgets data access to the outside world, in this way internal system details are encapsulated. The API is aimed at hiding the complexity of the underlying storage system and makes it accessible to developers or additional external modules.

### 2.4.9 Usage Guide

The Read API and Analytics modules aim at providing the main interface for querying, collecting, aggregating and analysing the data offered by the platform. Generic users meet the representation of the financial data and can use these modules for exploring the datasets and the information stored in the DataStore. A documentation for the API methods is provided to the users of the platform and the developers.

## 2.5 Visualization Module

The Visualization module is responsible for creating graphical representations of the data stored in the OpenBudgets.eu platform. It is capable of running complex queries on the DataStore through the API, or the data endpoint, and allows for the interpretation of the extracted data with different visualization components and techniques. Hence, it is directly connected to the DataStore module and the Read API.

### 2.5.1 Interface Identity

The Visualization module requires the data transfer interface from the DataStore module and read API module. This allows for data read access to the platform's stored datasets. Additionally it provides an interface for the selectable views components.

### 2.5.2 Resources Provided

The Visualization module provides different representations of the data provided. Many different type of visualizations can be created using the provided dimensions from the data representation. Currently the data is represented using two basic dimensions: date, and classification. Based on these dimensions, aggregates can be performed to create visualization data using logical operators.

### 2.5.3 Locally Defined Data Types

In this module, locally defined data types are not defined.

### 2.5.4 Error Handling

Error handling is managed by the Read API and DataStore modules. The visualization components utilized or plugged into this module are capable of handling errors independently.

### 2.5.5 Variability Provided

Variability of the visualizations depends on the provided dimensions in the data. Currently, temporal and categorical dimensions are envisioned, but additional dimensions could be implemented. This provides an adequate level of flexibility for possible modifications and updates.

### 2.5.6 Quality Attribute Characteristics

The Visualization module does not define any quality attribute in relation to performance for querying and accessing the data. This depends on the types of visualizations and visualization components adopted.

### 2.5.7 What is the Element Requirement

The Visualization module requires the DataStore and the Read API modules in order to create visual representational resources. These modules are the sources for the data used in the visualizations.

### 2.5.8 Rationale and Design Issues

A standalone independent visualization module is required in order to have a separation between the data model and its visual representation. This way the visualization of the data does not depend directly on the implementation of the data model (which is evolving over time), but only on the interface acting as a bridge.

### 2.5.9 Usage Guide

The visualization module is designed to provide easy to understand information and automated reporting features for non-expert users. At the same time, detailed information for experts who analyse the financial data more in detail is offered by the platform.

# 3 References


- Bachmann F. , Bass L. Et al. June 2002, Documenting Software Architectures: Documenting Interfaces
- OpenSpending Architecture, Retrieved from http://community.openspending.org/next/