DEVELOPMENT OF KNOWLEDGEBASE

D19 (SOFTWARE)

IDEALVis Consortium

http://idealvis.inspirecenter.org/

European Union
European Regional Development Fund

Republic of Cyprus

Structural Funds
of the European Union in Cyprus

ΙΔΡΥΜΑ ΕΡΕΥΝΑΣ ΚΑΙ ΚΑΙΝΟΤΟΜΙΑΣ

Executive Summary

This deliverable provides an overview of the KnowledgeBase, which is one of the fundamental layers that make up the IDEALVis platform architecture. The KnowledgeBase serves as a centralized repository that holds various types of data (i.e., generated information, information models, monitoring data and analytic algorithms) and further communicates those with various cooperating services e.g., Adaptation Engine. More specifically, this deliverable provides an overview of the KnowledgeBase by (i) providing an in-depth view of the platform's underlying database, and (ii) by presenting the architecture and design of (a) the Data Service which enables data querying, manipulation i.e., filtering and aggregation for transforming low-level data to high quality information; and (b) the Analytics Service which further transforms queried data using analytical algorithms into reports that can be visualized.

Table of Contents

List of Figures

1 Introduction

1.1 Deliverable Scope

This deliverable will focus on providing the architecture and the design of the KnowledgeBase layer found at the core of the IDEALVis platform. We will use references to code, XML/JSON configuration files and present visual diagrams (e.g., database diagrams) to illustrate the inner operations and structure of the various components (i.e., Database, Data Service and Analytics Service) under the layer of interest. The architecture and design principles applied during the development of the KnowledgeBase ensure that each component has high modularity and maintainability, while the same time preserving the lowest possible coupling between software components (i.e., changes and extensibility are feasible).

1.2 Deliverable Structure

The reset of the deliverable is structured as follows:

- Section 2 provides an overview of the system's database.
- Section 3 introduces the Data Service and describes the inner operations that allow for dataset querying.
- Section 4 introduces the Analytics Service and demonstrates how it consumes query results to produce analysis reports.
- Section 5 concludes the deliverable.

2 System Database

The project's knowledge base is built using Microsoft's SQL Server[1] database server and data management operations are performed using Transact SQL[2]. The platform was developed using ASP.NET Web APIs[3] and Entity Framework Core[4], an open-source and cross-platform object-relational mapper (O/RM), which enables us (using database migrations[5]) to interact and build the database by defining .NET objects in the C# language.

The current knowledgebase design is illustrated by the diagram in Figure 1, depicting only the most prominent tables of the database and their relationships. The design is user centric, allowing all interactions to be performed through the users table (AspNetUsers). Note that some tables presented to the left of the diagram in Figure 1 do not establish relationships with other tables. In the next sub-section, we provide a short description for each database table to further facilitate the readers understanding of the schema in Figure 1.

[1] https://www.microsoft.com/en-cy/sql-server/sql-server-downloads
[2] https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15
[3] https://dotnet.microsoft.com/apps/aspnet/apis
[4] https://docs.microsoft.com/en-us/ef/core/
[5] https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli

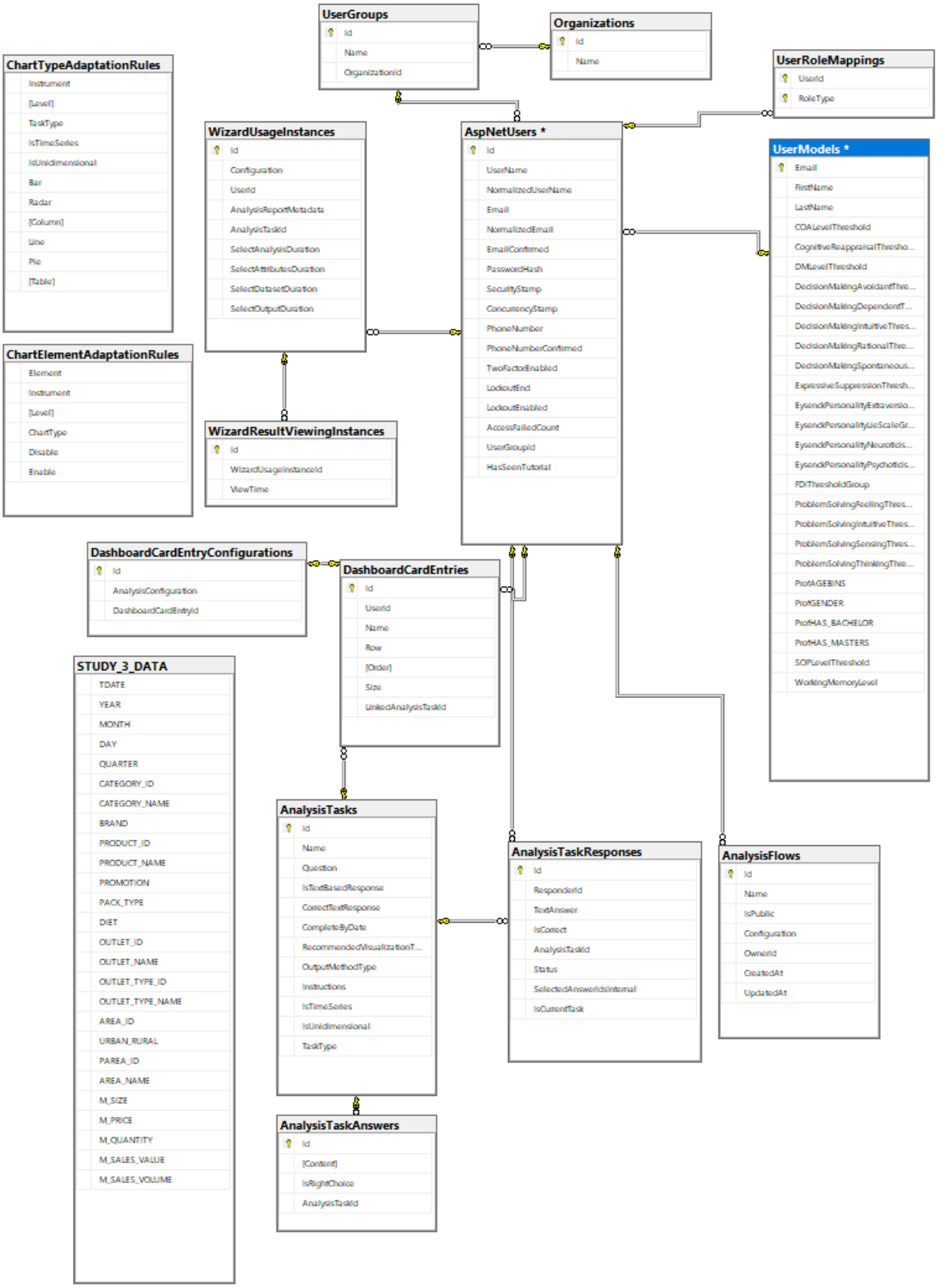

Figure 1 - IDEALVis Database Design

2.1 KnowledgeBase Tables

- **AspNetUsers:** Stores user's information such as profile and authentication information.
- **UserRoleMappings:** The role for each user is stored in this table.
- **UserModels:** Stores the user model for each of the users.
- **UserGroups:** This table maintains the user's group. A group represents a team within an organization (i.e., data preparation department). Groups are assigned to organizations (see below).
- **Organizations:** This table maintains the user's organization. An organization has many user groups.
- **WizardUsageInstances:** This table is used for storing the user's interactions when constructing analyses through the Analysis Wizard Interface (used for analysis tracking and pattern discovery). Each time a user creates or changes an analysis via the Analysis Wizard, the resulting analysis configuration and the time taken to perform every step of the Analysis Wizard are recorded in this table.
- **WizardResultViewingInstances:** This table is used for storing the user's view time with any analysis report (i.e., time spent gazing at a particular Analysis Wizard output). This data is used for analysis tracking and pattern discovery. Each time a user comes across an analysis report, this table is populated with a new time record related to a specific WizardUsageInstance that was used to create that report.
- **AnalysisFlows:** This table keeps all Analysis Wizard configurations (or analysis flows) that a user saved for later use.
- **DashboardCardEntries:** Dashboard cards (i.e., widgets) for every user are stored in this table. Information for every card regards the card's placement on the dashboard i.e., size and positioning. Moreover, a specific DashboardCardEntry is linked to a specific Analysis Task.
- **DashboardCardEntryConfigurations:** This table keeps a JSON query specification that defines the contents (i.e., analysis report) of each dashboard card.
- **AnalysisTasks:** This table keeps all the tasks that data analysts within an organization must solve. Tasks are defined by an Executive Data Analyst user.
- **AnalysisTasksAnswers:** For each analysis task that can be addressed using one of multiple answers (i.e., multiple choice task), this table keeps all the possible task answers, including an indicator to the correct answer.

- **AnalysisTaskResponses:** This table keeps all analysis task responses provided by users for each task. Moreover, this table marks which tasks are assigned to which users.
- **ChartTypeAdaptationRules:** All the adaptation engine rules that are used for selecting the best fit data visualization type.
- **ChartElementAdaptationRules:** All the adaptation rules that are used for selecting different visual element adaptations to be applied on the best fit data visualization.
- **<EXTERNAL_DATA>:** Any data that are used for exploration and analysis purposes are imported into the database as an external table (e.g., STUDY_3_DATA).

2.2 Analysis Datasets

The Data Service supports both locally stored data (see <EXTERNAL DATA> in the previous section) or connecting to datasets that are maintained on external to the platform sources/servers. These datasets are the ones which users explore for addressing the analysis tasks assigned to them.
To support the studies, we opted for the former option (i.e., to include the analysis dataset in the platform's database) for ensuring a smoother and seamless pilot study deployment.

3 Data Service Layer

Data retrieval and analysis in IDEALVis is performed using a graphical user interface coined "Analysis Wizard", which enables a data analyst user to request data for a specific analysis, without explicitly having to type a query (i.e., code or expressions of a specific query language). Instead, the Analysis Wizard enables the user to easily construct a database-agnostic JSON query specification for a given analysis, using buttons and drag and drop functionality. Moreover, the Analysis Wizard is powered by the Query Engine, which is responsible for three key operations including: (i) storing and parsing XML files that describe each dataset that the Query Engine/Analysis Wizard can interact with; (ii) interpreting and transforming the JSON query specification to valid/realized query language (e.g., Microsoft Transact SQL); and (iii) executing the realized query on the target dataset, retrieving the results and returning them as a list of dictionaries i.e., JSON format. The sections below provide the specifications of the XML dataset descriptors, and further describe the process of transforming a JSON query specification to a query string.

3.1 Dataset XML Descriptors

The Query Engine is agnostic of how the data are represented (i.e., their schema). Therefore, each dataset that the platform interacts with needs to be described using a specific XML descriptor file. For a given dataset, the descriptor file provides information, such as (i) the dataset name, (ii) the dataset available schema/attributes, their data types and their human-readable name i.e., label, and (iii) the available attribute hierarchies. An example sales dataset descriptor file can be seen at Figure 2. More specifically, the descriptor aims to provide all the necessary dataset metadata and facilitate a mapping between the actual database constructs and the internal Query Engine representations. This mapping is achieved by the ***db*** attribute on the and ***tdata*** XML elements as seen in Figure 2.

```
<transactional_data id="transactions" db="STUDY_3_DATA" label="Study 3 Data">
    <data>
        <tdata id="date" type="attribute" db="IDATE" label="Date" format="date" />
        <tdata id="year" type="attribute" db="YEAR" label="Year" format="int" />
        <tdata id="month" type="attribute" db="MONTH" label="Month" format="int" />
        <tdata id="day" type="attribute" db="DAY" label="Day" format="int" />
        <tdata id="quarter" type="attribute" db="QUARTER" label="Quarter" format="int" />
        <tdata id="brandName" type="attribute" db="BRAND" label="Brand" format="string" />
        <tdata id="productName" type="attribute" db="PRODUCT_NAME" label="Product" format="string" />

        <tdata id="mSalesValue" type="measure" db="M_SALES_VALUE" label="Sales Value (Measure)" format="decimal" />
        <tdata id="mSalesVolume" type="measure" db="M_SALES_VOLUME" label="Sales Volume (Measure)" format="decimal" />
    </data>

    <hierarchies>
        <hierarchy id="dateHierarchy" label="Date Hierarchy">
            <level tdataid="year">
                <level tdataid="month">
                    <level tdataid="day" />
                </level>
            </level>
        </hierarchy>
    </hierarchies>

</transactional_data>
```

Figure 2 – Example Sales Dataset Descriptor File

Given a descriptor XML file, the Query Engine parses it down to a ***DatasetSpec*** C# object which is used for communicating dataset specifics to the front-end (i.e., details about available datasets and their attributes etc. used to populate information in the Analysis Wizard interface) and for mapping/transforming the JSON query specification into a proper query language (i.e., given that the query spec has the mSalesValue measure in the list of selected attributes this would map to M_SALES_VALUE in the actual query language as dictated by the example XML descriptor in Figure 2.

3.2 JSON Query Specification

While the user is interacting with the Analysis Wizard interface, all user's choices are combined into the JSON query specification. For example, in order to retrieve the desirable data for an example task narrated as "*Identify the month with the highest sales volume during 2019 for brand IdealCola*", the Analysis Wizard attributes selection step will need to be configured as illustrated in Figure 3 and Figure 4. The illustrated configuration will result to the generated JSON query specification as shown in Figure 5, assuming that (i) the transactions dataset (Figure 2) was selected along with the list report analysis and a line chart for output.

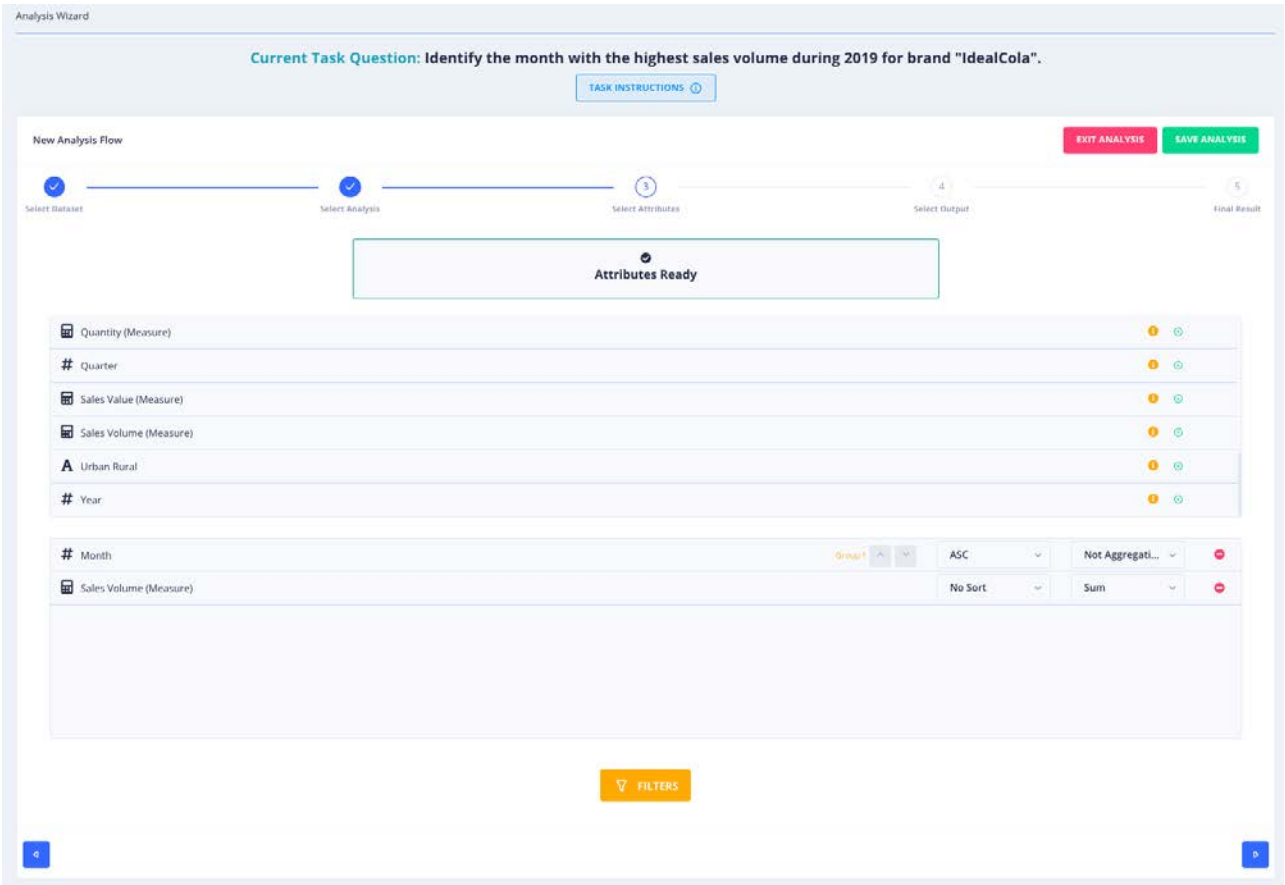

Figure 3 - Analysis Wizard - Selecting Attributes

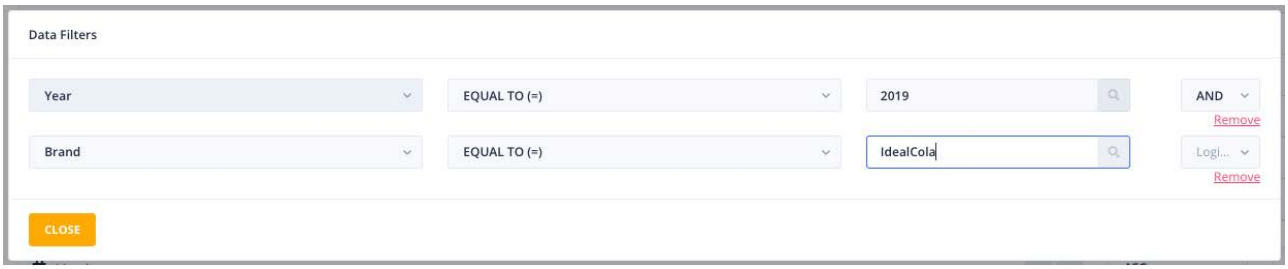

Figure 4 - Analysis Wizard - Setting Filters

```
{
  "datasetId": "transactions",
  "analysisMethod": "ListReport",
  "outputMethod": "Line",
  "attributes": [
    {
      "id": "month",
      "aggregationType": null,
      "type": "Categorical",
      "order": "Ascending"
    },
    {
      "id": "mSalesVolume",
      "aggregationType": "Sum",
      "type": "Numerical",
      "order": null
    }
  ],
  "filters": [
    {
      "property": "year",
      "compareValue": "2019",
      "relationalOperator": "EqualTo",
      "logicalOperator": "AND"
    },
    {
      "property": "brandName",
      "compareValue": "IdealCola",
      "relationalOperator": "EqualTo",
      "logicalOperator": null
    }
  ]
}
```


Figure 5 - Example JSON Query Specification

Once the user has finished configuring their analysis the fifth step of the Analysis Wizard takes the generated JSON query specification and sends it to the server so it can be realized to a valid query language by the Query Engine. The Query Engine implementation is flexible and can easily be extended to support many query languages. For the purposes of this project, the current implementation allows only for realizing the JSON query specification to Microsoft's Transact SQL. For instance, given the above JSON query specification as input, the Query Engine will produce the following SQL Query as shown in Figure 6.

```
SELECT [S0].[MONTH] AS [C0], SUM([S0].[M_SALES_VOLUME]) AS [C1]
FROM [STUDY_3_DATA] AS [S0]
WHERE [S0].[BRAND] = 'IdealCola' AND [S0].[YEAR] = 2019
GROUP BY [S0].[MONTH]
ORDER BY [C0] ASC
```

Figure 6 - Example Realized SQL Query

4 Analytics Service

The Analytics Service is responsible for further analyzing and preparing queried data retrieved from the dataset (using the Data Service), for the purpose of producing meaningful results according to the request of the user. Specifically, this service exposes multiple knowledge data discovery mechanisms (e.g., statistical algorithms for descriptive, predictive, and prescriptive analytics) that can be executed on the queried data. Acting as a pipeline, this service ingests the data that were requested from the dataset using the Data Service and transforms those data according to the analysis algorithm that was selected. If for example the Forecasting Analysis was selected, the queried data will be fitted to the forecasting algorithm that will further predict future values of the data distribution according to a set of parameters provided from the user as required by the algorithm.

During the process of constructing an analysis (i.e., JSON query specification) the user is required to select an analysis method to run on the queried data. This action is performed at step 2 of the Analysis Wizard as seen in Figure 7. The analysis method selected by the user is always reflected on the JSON query specification (Figure 5) as shown on line 3.

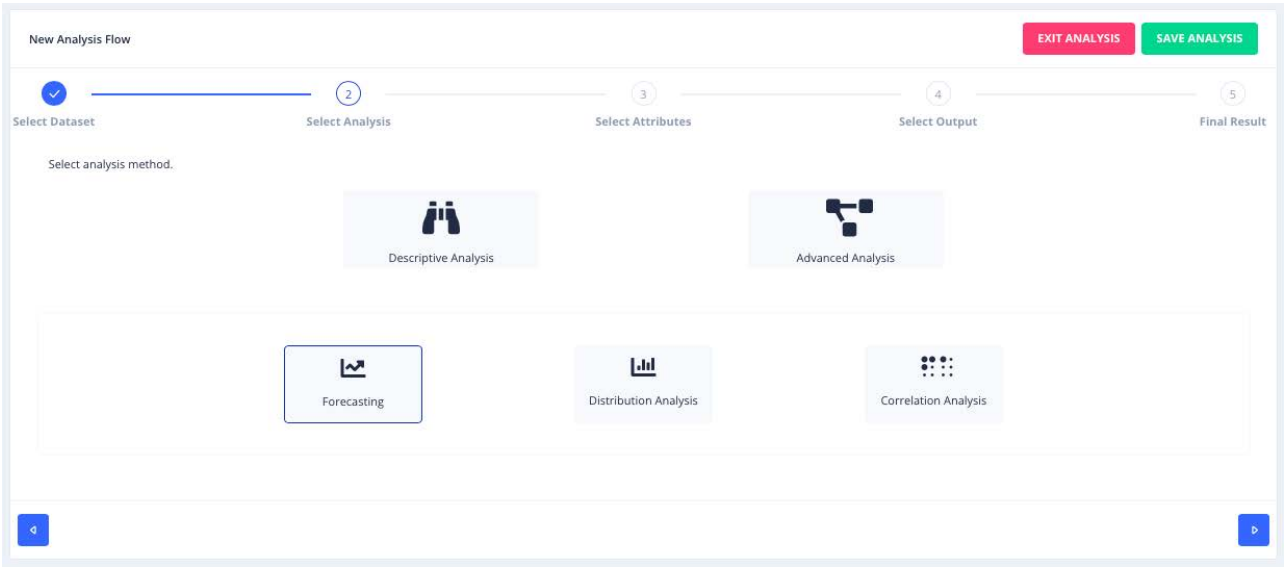

Figure 7 - Analysis Wizard – Selecting Analysis

In the next sub-sections, we provide the specifications of how an analysis is defined in the Analytics Service by illustrating parts of the codebase. Moreover, using diagrams we demonstrate the flow of information from ***query*** to ***visualization*** in order to further facilitate our explanation of the analytics supported by the KnowledgeBase.

4.1 Analysis Definition

Every analysis in the system is represented by an analysis method class which contains all the analysis logic. Data analyses in the system are implemented in an open manner, using a flexible hierarchical and polymorphic structure, which enables the KnowledgeBase to be easily extensible to a new set of analyses, while also being easily maintainable and testable. For extending the current system with a new analysis method, a new analysis method class needs to be created by inheriting the **AnalysisBase** abstract class and implement all required methods accordingly. A list of all the analysis methods implemented in the KnowledgeBase are presented in Figure 8. Below

we describe the **AnalysisBase** abstract class and the methods/functions that each analysis method class has to implement in order to satisfy the requirements of the abstract class.

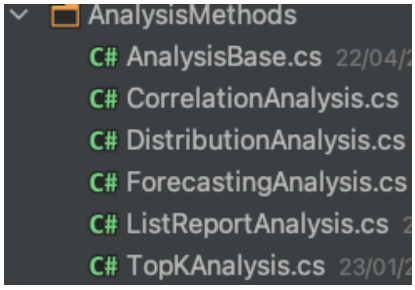

Figure 8 - Implemented Analysis Methods

AnalysisBase: Represents the actual analysis class that contains the necessary procedures that run when the user initiates a specific analysis. When this class is inherited for the purpose of creating a new analysis, the programmer is required to provide definitions to the three methods described below:

- **ProvideAttributeValidator:** Every analysis method has its own attribute validator which specifies the minimum and maximum data attributes and attribute data types required for the specific analysis to run. Moreover, a validator dictates which data visualizations can be used as the output for the specific analysis method. This method enables the programmer to assign an attribute validator object to the specific analysis method. Moreover, the validator object is run before the analysis query to make sure that all attributes were provided by the user.
- **ExecuteQuery:** This method is where the actual Query Engine is called for the purpose of retrieving the data required for the analysis. Each specific analysis method can define its own way of querying data via this method.
- **RunAnalysis:** This method is where the resulting query data are transformed and analysed according to the selected analysis method. In this function the programmer is required to build and return the **AnalysisReport** object resulting from the analysis.

Moreover, the AnalysisBase abstract class has a few pre-built methods, which are used across all analysis method classes as required. The user can also overwrite those methods if the need arises. Some of those methods are described below:

- **FromListOfDictionariesOfObjectKeyToStringKey:** This method performs a transformation on the data returned by the Query Engine. Essentially this method transforms the data from a list of dictionaries of key type object and value type object to a list of dictionaries of key type string and value type object.
- **ApplyDataToOutputMethodType:** This is an essential method which takes the final set of data resulting from the analysis and packs them in an **OutputMethodInstructions** object which contains all the information on how the data is to be visualised. The generated

OutputMethodInstructions object is always the result of a specific output method (i.e., data visualization) instructions builder, who dictates how the data are to be connected on the data visualization selected by the user. Moreover, if adaption is enabled for the current analysis task the adaptation engine will be called accordingly from this very method. Similar to output method builders the adaptation engine also returns an **OutputMethodInstructions** object.

- **BuildGroupByReportTitle:** This method automatically builds a default title for the resulting analysis report based on the data grouping provided.

Moreover, the AnalysisBase abstract class has a number of properties which are essential for every analysis method. Some of the essential properties are listed below:

- **analysisSemantics:** This property is an object which keeps various details regarding the user's request. This object is essentially built by the Analytics Service for all analyses prior to invoking the actual analysis object for data processing. Moreover, the **analysisSemantics** object represents a deconstructed, more detailed version of the JSON query specification of the analysis request (e.g., contains number of categorical, or measure attributes, number of aggregations etc). Information found in this property are used for example when the analysis validator checks if all essential attributes were selected by the user**.**
- **outputMethodType:** This property represents the selected output method i.e., data visualization selected by the user for the current analysis.
- **analysisParameters:** This property is a specific object different for every analysis method that is responsible for keeping any extra parameters provided to the analysis by the user. For instance, for the Top-K analysis this object will contain the K value provided by the user.

Moreover, all analysis methods implemented in the system are required to produce a single output; the **AnalysisReport** object as mentioned previously in the **RunAnalysis** method. This object's properties are described below.

- **ReportTitle:** Makes up the title of the analysis report that provides insight on to what is being analysed or visualized.
- **OutputMethodInstructions:** This is the object that contains all the analysis transformed data along with specific data visualization instructions as to how these data should be rendered by the data visualization engine.

4.2 Rendering Analysis Reports

Every new analysis added to the system, is enabled and accessible by data analysts only once it has been registered on the Analytics Service as well as through the appropriate Service Manager endpoint. Moreover, it must be noted that all resulting analysis report objects are rendered at the last step of Analysis Wizard. The Analysis Wizard which runs on the client's browser hosts the data visualization engine that can receive any given **AnalysisReport** object and render it accordingly. The diagram in Figure 9 further illustrates the flow of information that takes place when the data analyst user issues a specific analysis request via the IDEALVis platform.

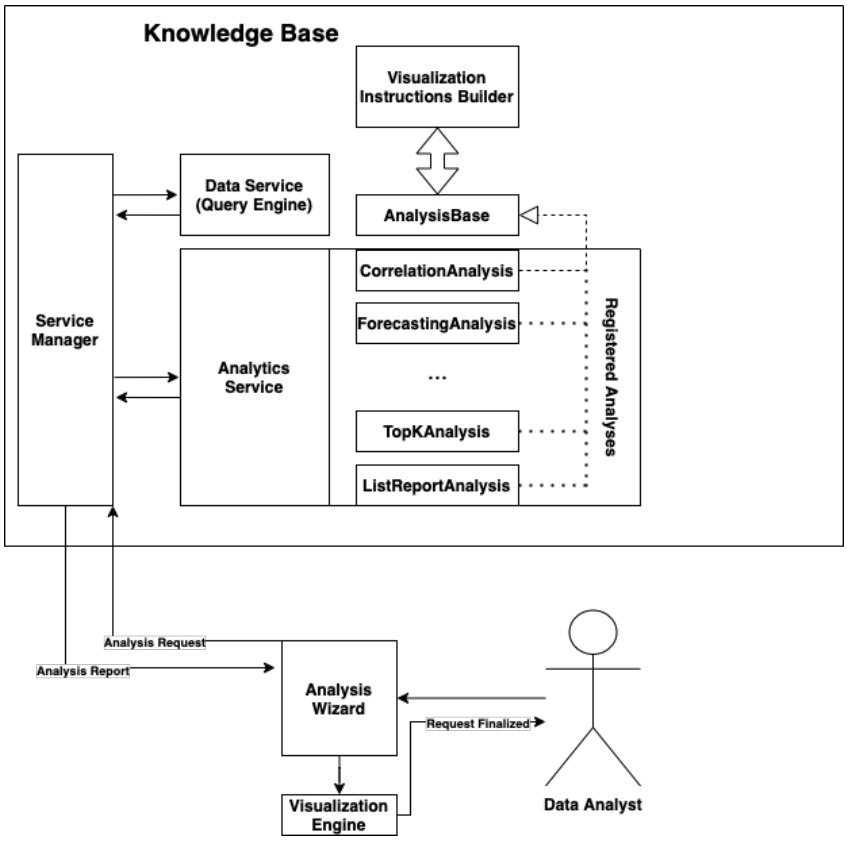

Figure 9 - Overall Analysis Request Data Flow

5 Conclusions

This deliverable presented the overall architecture and purpose of the KnowledgeBase and further described its underlying services, components and their interaction with the Analysis Wizard and Adaptation Engine. More specifically, the deliverable (i) illustrated the various database tables found on the platform's database and provided a brief description of how each table allows for the proper platform operation, (ii) demonstrated the Data Service that enables the platform to connect on external datasets using XML descriptor files and execute queries using JSON query specifications that are constructed by the user via the Analysis Wizard, and finally (iii) presented the Analytics Service and how an analysis method is defined in the system, while further explaining how the resulting report of an analysis request is generated. Moreover, we specified at which point of the analysis the Adaptation Engine can intervene. Finally, we have illustrated the overall analysis flow of how the user's request becomes an analysis report of output method instructions that can be rendered into a data visualization by the visualization engine.