DESIGN AND DEVELOPMENT OF THE SERVICE MANAGER

D17 (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 Service Manager, which is one of the main platform components, responsible for handling the bidirectional communication between software components. More specifically, the main purpose of the Service Manager is to expose the functionality of all components residing in the IDEALVis platform, by providing appropriate Application Programming Interfaces (APIs). IDEALVis aims to bring high flexibility and transparency between various software components by adopting the Service-Oriented Architecture (SOA) paradigm, thus, the deliverable focuses on how the Service Manager implements the SOA paradigm, how it makes available, orchestrates, and interconnects different services, including how other system components leverage those services for delivering the adapted data visualization experience to the end user.

Table of Contents

List of Figures

1 Introduction

IDEALVis, as a software platform, is designed with the objective to meet several software qualities, including performance, correctness, robustness, and reliability. As such, the IDEALVis consortium has decided to follow the Service-oriented Architecture (SOA) paradigm to meet these qualities and support modularity, expandability, and interoperability. SOA encapsulates specific application logic in services (stand-alone entities) that interact with the internal platform's components and with external systems, via uniformly defined interfaces that expose available methods or actions which a client system can execute on the service.

The list below indicates a number of benefits that SOA offers to IDEALVis:

- **Reliability:** Having small and independent services enables easier debugging and testing of separate features rather than having to debug massive code all at once i.e., all the platform code.
- **Loosely Coupled:** SOA is a loosely coupled architecture (i.e., encourages the development of independent services for higher efficiency) inspired by object-oriented programming design that aims to reduce coupling between classes for minimising the risk of breaking class relationships when a specific class is updated.
- **Maintainable:** Each service is an independent unit that can be updated, maintained, tested etc. in isolation without affecting other services, always assuming that the service implements the publicly exposed interface.
- **Reusability:** A platform that is built with SOA in mind, consists of small, self-contained, and loosely coupled services, therefore, reusability of those services is possible withing the system itself or across other systems.

1.1 Deliverable Structure

The rest of the deliverable is structured as follows:

- Section 2 (i) describes the Service Manager's underlying services, (ii) presents all the low-level Application Programming Interfaces (APIs) for each service exposed by the Service Manager, and (iii) finally demonstrates how the Service Manager exposes data access to other related components.
- Section 3 concludes the deliverable.

2 The Service Manager

The Service Manager (illustrated in Figure 1) enables other platform's components to seamlessly access all services, including services residing in the KnowledgeBase layer, via appropriate APIs. In particular, the Service manager provides access to: (i) data ingestion and pre-processing mechanisms; (ii) data manipulation and querying mechanisms, residing in the KnowledgeBase; and (iii) data analysis algorithms (e.g., statistical algorithms for descriptive and predictive). Furthermore, the Service Manager exposes the following services to different platform components:

- **User Model Retriever Service**
- **Data Service**
- **Analytics Service**
- **Analysis Tracker Service**

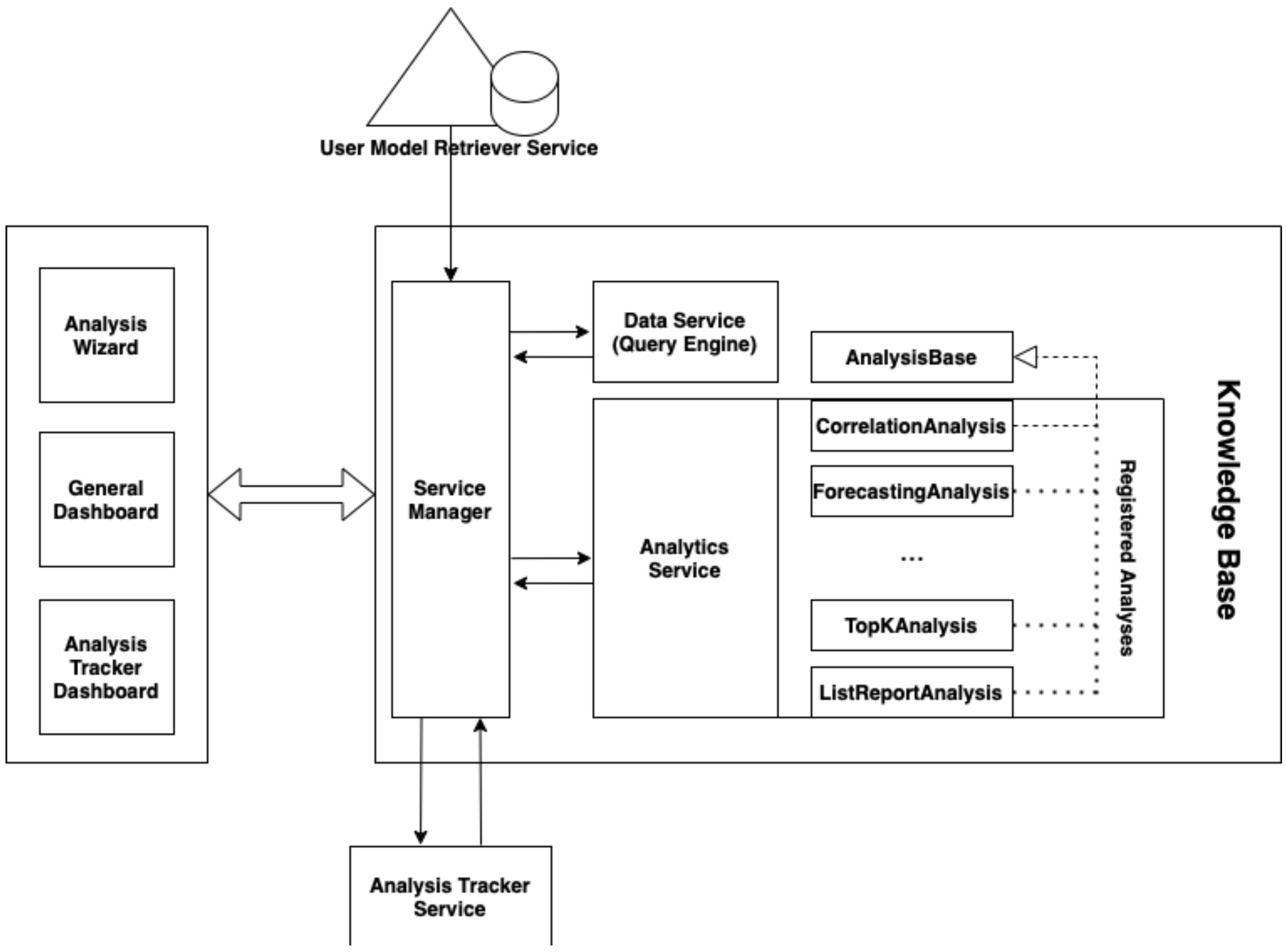
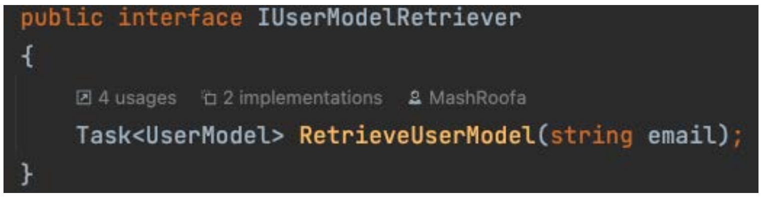


Figure 1 – The Service Manager

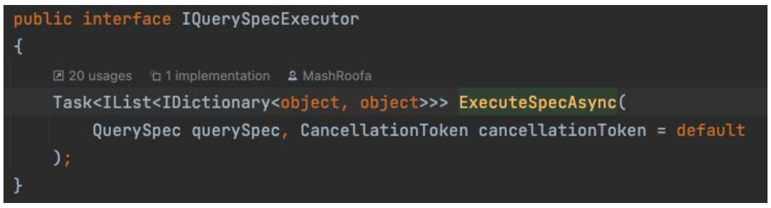
2.1 User Model Retriever Service

The user model is the central repository that maintains the different user characteristics (e.g., psychometric indicators, demographic data) for each of the users. This is an essential component required by the platform to produce adaptive data visualisations according to the user's characteristics. Specifically, the user model is exposed to the different platform components via the Service Manager through a service called **UserModelRetriever**. This is a standalone service that the Service Manager utilises for enabling access to the users' models. Additionally, since IDEALVis follows the SOA approach, the **UserModelRetriever** service was built with flexibility in mind enabling the platform to seamlessly access user models which either reside in the local platform's database, or even externally i.e., it can access user models which are hosted by external sources/servers, decoupled from the IDEALVis core platform. Furthermore, the Service Manager enables other platform components and services to seamlessly access and query the active user's user model via a simple programming interface as seen in Figure 2. This allows external platforms to access the user's model by invoking the **RetrieveUserModel** method via the Service Manager, given that they have been granted appropriate access rights. Any service requesting a user model via the Service Manager is not aware whether the user model is locally (i.e., in the platform's database) or externally hosted, since the Service Manager encapsulates this logic behind the interface seen in Figure 2. More specifically, the user model service provides two classes which implement the interface in Figure 2 for achieving access to internal or external to the platform user models. Those classes are the **LocalUserModelRetriever** and the **RemoteUserModelRetriever**. For the purposes of this project and during the pilot study we utilised the local user model retriever since the user models of the users were hosted in the actual IDEALVis database for performance reasons.


```
public interface IUserModelRetriever
{
    4 usages   2 implementations   MashRoofa
    Task<UserModel> RetrieveUserModel(string email);
}
```
Figure 2 - User Model Retriever Interface

2.2 Data Service

The Data Service makes an essential part of the IDEALVis platform, as it is responsible for executing queries on the underlying dataset used for analysis. This service exposes a number of methods/actions which allow for (i) querying a transactional dataset that is loaded with an appropriate database connection string and a metadata file which describes the dataset attributes; (ii) performing aggregations, grouping, and filtering on selected data of the dataset; and for (iii) retrieving statistics for a selected dataset or specific dataset attributes. The Data Service is exposed by the Service Manager to the Analytics Service and to the Analysis Wizard interface. Specifically, the Analytics Service utilises the Data Service through Service Manager for the purpose of processing the user analytical request and returning the data that is to be fed in the underlying analysis method requested by the user, as seen in deliverable D19. The Service Manager exposes the possibility to execute queries to the Analytics Service via the following interface as seen in Figure 3.


```
public interface IQuerySpecExecutor
{
    20 usages   1 implementation   MashRoofa
    Task<IList<IDictionary<object, object>>> ExecuteSpecAsync(
        QuerySpec querySpec, CancellationToken cancellationToken = default
    );
}
```
Figure 3 - Data Service Execute Query Interface

The Data Service utilises inheritance and polymorphism to support openness and extensibility. Currently, this service is able to process/parse a JSON query specification produced via the

Analysis Wizard interface and translate it to an appropriate SQL query, which later is executed against the underlying dataset. The Data Service can produce ISO/IEC 9075:2016 SQL statements; however, there are specific functions (e.g., windowed functions, ranking functions) that are implemented in T-SQL and are thus supported by Microsoft's SQL Server. Despite these functions, the flexible approach taken when designing the Data Service makes it possible to quickly adapt the service and translate the JSON query specification in other variants of SQL (or other standards) which are supported by other database servers e.g., PostgreSQL. Moreover, the design of this service makes it possible to access both datasets which are local to the platform's database or datasets which are externally hosted on another database server.

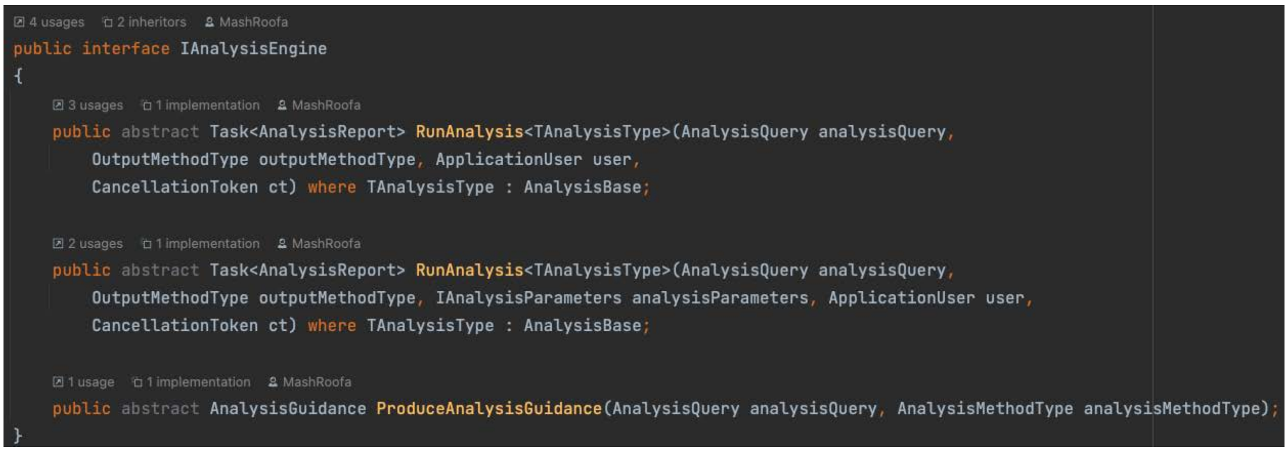
The Service Manager also exposes several other services which enable the direct interaction of the Data Service to the Analysis Wizard. Those services essentially, provide useful information to the user prior to running a specific data analysis. Some of this information for instance includes descriptive statistics for the attributes of the dataset, available values for filtering a specific data attribute etc. Below we list some of the most important Data Service methods exposed to the Analysis Wizard via the Service Manager:

- **GetDatasetModelsByIdAsync:** This method provides a list of all the available datasets that the Data Service can query from. Moreover, this method provides information about all the available data attributes of a given dataset.
- **GetDistinctCategoryValues:** Given a specific dataset and a categorical data attribute that resides inside that dataset, this method will provide a list of the unique attribute values.
- **GetRowCount:** Given a specific dataset this method will count the number of rows which the dataset currently has.
- **GetDatasetSummary:** Given a specific dataset this method will produce descriptive statistics regarding each of the dataset's attributes.
- **GetCategorySummary:** Given a specific dataset and a categorical data attribute that resides inside that dataset, this method will produce descriptive statistics regarding this attribute.
- **GetMeasureSummary:** Given a specific dataset and a numerical data attribute that resides inside that dataset, this method will produce descriptive statistics regarding this attribute.
- **GetCategoryMode:** Given a specific dataset and a data attribute, this method will return the mode value of the data attribute.

2.3 Analytics Service

The Analytics Service enables the analysis of data by exposing several knowledge discovery mechanisms (e.g., statistical algorithms for descriptive and predictive analytics) that can be applied on the data requested by a user. Essentially, the Service Manager in this case facilitates the communication between the Data Service and the Analytics Service, as disused previously in Section 2.2. This communication takes place via the interface presented in Figure 3. The Analytics

Service ingests data that were requested from the dataset using the Data Service and processes them according to the selected analysis algorithm. Once the algorithmic operation is completed, the processed output is sent back to the Service Manager and subsequently back to the Analysis Wizard. The Analytics Service can be easily enhanced with new analysis methods as this was presented in D19.
Figure 4 presents the Analytics Service interface which is used by the Service Manager to expose the analytics functionality to other platform components (e.g., Analysis Wizard, Dashboard). This interface essentially enables the Service Manager to delegate analysis related requests to the Analytics Service.



```
4 usages   2 inheritors   MashRoofa
public interface IAnalysisEngine
{
    3 usages   1 implementation   MashRoofa
    public abstract Task<AnalysisReport> RunAnalysis<TAnalysisType>(AnalysisQuery analysisQuery,
        OutputMethodType outputMethodType, ApplicationUser user,
        CancellationToken ct) where TAnalysisType : AnalysisBase;

    2 usages   1 implementation   MashRoofa
    public abstract Task<AnalysisReport> RunAnalysis<TAnalysisType>(AnalysisQuery analysisQuery,
        OutputMethodType outputMethodType, IAnalysisParameters analysisParameters, ApplicationUser user,
        CancellationToken ct) where TAnalysisType : AnalysisBase;

    1 usage   1 implementation   MashRoofa
    public abstract AnalysisGuidance ProduceAnalysisGuidance(AnalysisQuery analysisQuery, AnalysisMethodType analysisMethodType);
}
```

Figure 4 - Analytics Service Interface

Below we explain the different methods and method parameters, offered through the Analytics Service interface:

RunAnalysis: This overloaded method is one of the key entry points of the system where the query of the user is initially submitted. All the analysis settings and the user-defined query are used as input to this specific method. Note that the JSON query specification (i.e., AnalysisQuery parameter) is passed to the Data Service for execution at a later point by the actual analysis method used. Once the analysis method finished execution it returns an **AnalysisReport** object (described in deliverable D19) which is used by the visualization engine for rendering the final visual report (last Analysis Wizard step). Below, we describe each of the parameters passed to this method:

- **<TAnalysisType>:** This generic parameter signifies which type of analysis is to be executed by the system. This parameter can receive any object type that inherits the **AnalysisBase** abstract class, which is the class used to define new analysis methods for the Analytics Service. The concept presented here demonstrates the flexibility of the platform to support new types of analyses without having to modify the actual Service Manager communication methods / interface.
- **AnalysisQuery:** The JSON query specification constructed via the analysis wizard is initially, transformed into this **AnalysisQuery** object which is the object required by the Analytics Service for further processing. Note that before the analysis method calls the Data Service

the **AnalysisQuery** object is modified according to the needs of the analysis and then it is further transformed into a **QuerySpec** object as required by the Data Service for execution.

- **OutputMethodType:** This parameter specifies the type of data visualization to be used for the analysis output. Moreover, this parameter dictates if an adapted data visualization will be used or not.
- **IAnalysisParameters:** 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.
- **ApplicationUser:** The information about the user performing the analysis. This object is used later for retrieving the user's code needed for requesting the user model from the User Model Retriever Service.

As an additional note to the **RunAnalysis** method it must be mentioned that all analyses pinned on the user's General Dashboard are actually using the **RunAnalysis** method from the Service Manager in order to execute the analysis and render their results, similar to how the last step of the Analysis Wizard calls the Service Manager for executing a user defined analysis.

ProduceAnalysisGuidance: 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. The **ProduceAnalysisGuidance** method is called by the Analysis Wizard periodically (i.e., every time the user makes a change to the set of analysis attributes) and is used to validate the inputs the user provides to the analysis. Specifically, this method takes the current version of the JSON query specification and also the type of the analysis method the user is trying to perform. Based on those inputs this method executes the corresponding analysis validator and checks if the user's inputs are valid. Accordingly, this method returns the **AnalysisGuidance** object which packs a set of instructions to further guide the user on how to proceed with setting up all inputs to the analysis method. The parameters to this function are:

- **AnalysisQuery:** A JSON query specification constructed via the analysis wizard. This is submitted prior to running the analysis for inspection by this method.
- **AnalysisMethodType:** This is an object that specifies the type of analysis method the user is trying to construct using the Analysis Wizard.

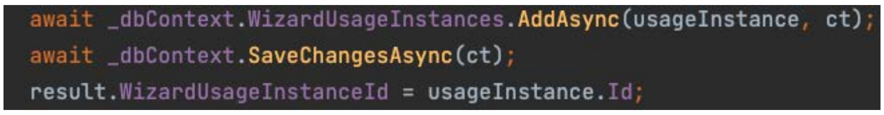
2.4 Analysis Tracker Service

Analysis tracking in IDEALVis is performed using a set of custom-made client-side tracking mechanisms which record the user interaction, such as constructing an analysis via the Analysis Wizard and viewing a specific data analysis report i.e., rendered data visualization. Moreover, those tracking records are prepared, analysed, and presented to the user through the Analysis

Tracker Dashboard where the user can reflect on their efficiency and effectiveness when they are preforming data explorations using IDEALVis. The Service Manager exposes two endpoints for the Analysis Tracker Service. The first endpoint purpose is the persistence of the client-side tracking data in appropriate database tables, while the second endpoint is used for providing the Analysis Tracker Dashboard with appropriate data regarding the user's analytical performance.

2.4.1 TRACKING RECORD PERSISTENCE

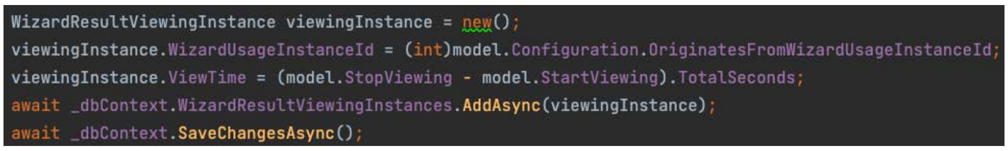
Analysis Wizard Tracking: While the user constructs an analysis via the Analysis Wizard a specific client-side procedure counts the number of milliseconds the user spends on each of the Analysis Wizard's steps. Moreover, once the user reaches the final Analysis Wizard step their analysis request is automatically submitted to the Service Manager along with the set of tracking information regarding every Analysis Wizard step. Moreover, once the request reaches the Service Manager two steps take place. Initially, the analysis request is sent to the Analysis Service and the set of tracking data are automatically persisted in the **WizardUsageInstances** database table. Once the analysis is executed the Service Manager returns the analysis report back to the user's browser for visualization. In addition to the analysis report the Service Manager also appends a wizard usage instance unique identifier on the analysis result (Figure 5). This identifier is added on the analysis report so we can later reference each analysis report found in the platform (e.g., a report might be pinned in the General Dashboard for instance) with a specific **WizardUsageInstance** which is required for when tracking the user's report viewing time as explained in the next paragraph.



```
await _dbContext.WizardUsageInstances.AddAsync(usageInstance, ct);
await _dbContext.SaveChangesAsync(ct);
result.WizardUsageInstanceId = usageInstance.Id;
```

Figure 5 - Adding Wizard Usage Unique Identifier on Analysis Result

Analysis Report Tracking: Every time the user is gazing at a particular analysis report produced by the system the client-side tracking mechanism records the time taken by the user before they take the analysis out of focus. Note, that analysis reports produced by the system may reside on the user's General Dashboard or at the final step of the Analysis Wizard. Moreover, once the user stops gazing at a particular analysis report, the tracker submits the tracking information to the Analysis Tracker Service via the Service Manager. The submitted data include the time (in milliseconds) the user was gazing at the particular analysis report, and also the report's **WizardUsageInstance** unique identifier. These data are handled by the Analysis Tracker Service and are persisted in the **WizardResultViewingInstances** database table (Figure 6)**.**

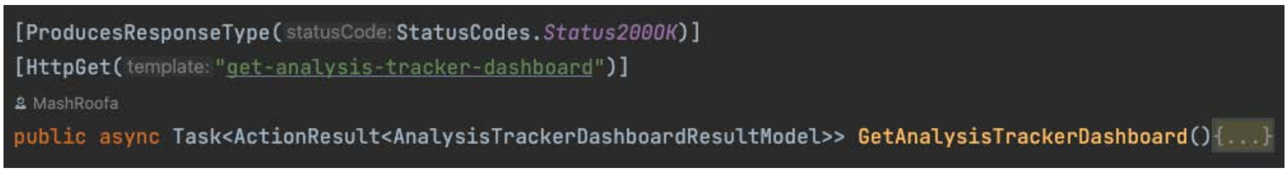


```
WizardResultViewingInstance viewingInstance = new();
viewingInstance.WizardUsageInstanceId = (int)model.Configuration.OriginatesFromWizardUsageInstanceId;
viewingInstance.ViewTime = (model.StopViewing - model.StartViewing).TotalSeconds;
await _dbContext.WizardResultViewingInstances.AddAsync(viewingInstance);
await _dbContext.SaveChangesAsync();
```

Figure 6 - Persisting Analysis Result View Time for Analysis Tracking

2.4.2 ANALYSIS TRACKER DASHBOARD

The Analysis Tracker Dashboard leverages data produced from the Analysis Tracker Service to provide the user with insights regrating their analytical performance. Essentially, the tracking data collected for the user rare aggregated, and several data visualizations are created on the Analysis Tracker Dashboard regarding the overall performance of the user but also the performance of the user per analysis task addressed. The Service Manager exposes the Analysis Tracker Service to the Analysis Tracker Dashboard via the endpoint presented in Figure 7.



```
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
[HttpGet( template: "get-analysis-tracker-dashboard")]
MashRoofa
public async Task<ActionResult<AnalysisTrackerDashboardResultModel>> GetAnalysisTrackerDashboard(){...}
```

Figure 7 - Analysis Tracker Dashboard Endpoint

3 Conclusions

This deliverable presented the overall architecture and purpose of the Service Manager and further described its underlying services. The rationale behind our choice for implementing the various services handled by the Service Manager using the SOA paradigm was justified, and a set of related advantages were discussed. Moreover, we presented the different system’s services including the User Model Retriever Service, the Data Service, the Analytics Service, and the Analysis Tracker Service. Additionally, we explained how those services utilize the Service Manager for intercommunicating with each other, and we also discussed how those services are exposed via the Service Manager to other system components using different interfaces, endpoints, and methods.