DREAM

# DD

Design Document

Alessandro Cecchetto (10865806)
Mattia Siriani (10571322)
Matteo Visotto (10608623)

Computer Science and Engineering
Software Engineering 2 course @ Politecnico di Miliano

v1.0 - 09/01/2022

# Contents

## Information

This project has been built following the indications of the stakeholders, referring to what is published in the official repository (https://github.com/UNDP-india/Data4Policy), taking into consideration, for some aspects, also the specification provided by the teachers in charge of the course.

Document version: v1.0 - 09/01/2022

# 1 Introduction

## 1.1 Purpose

This document is meant to provide a detailed explanation about the technical details needed to implement the Dream platform. In particular, the architecture will be presented alongside the modules and the interfaces that will compose the system.
Moreover, the functionalities offered by the software will be shown through the runtime view, highlighting the interaction and the message exchanged between the components.
Finally, the connection between the different interfaces will be presented and a dedicated section will outline the implementation, integration and test plan.

## 1.2 Scope

The application is divided in two different areas: The Forum Area and the Data Aggregator one. Both are visible for every Visitor and also the data are freely accessible, but advanced functions requires to be registered.

Registered Users have the permission to publish content on the forum and interact with the Policy makers in the discussions.

Policy makers are also the creator of the discussions in the forum and they act as moderator of the platform, accepting or declining post publication by user and also they can modify every post in the forum.
In addition, Policy makers can recalculate the ranking lists, through the Deviance algorithm by specifying the parameters set they are interested in.

Finally, in the Data aggregator area, the Administrator can manage the data sources provided on the platform.

## 1.3 Definition, acronyms, abbreviations

### 1.3.1 Definition

- **Client-side scripting:** code generated to be run on the client browser without the necessity of a server to be executed.
- **Code On Demand:** in a distributed scenario it is the procedure in which a client obtain a piece of software (executable code) by requesting it to a server.
- **RESTful:** it's a software architectural style that defines a set of constraints to be used for creating Web services.
- **Tier:** In general, a tier is a row or layer in a series of similarly arranged objects. In computer programming, the parts of a program can be distributed among several tiers, each located in a different computer in a network.

- **Web Interface:** it permits to use a service only through a Web Browser.
- **Load balancer:** it is a device that allows to balance the workload between servers, to maintain their capacity at an optimal level.
- **Assertion Consumer Service:** An Assertion Consumer Service is SAML terminology for the location at a ServiceProvider that accepts response messages (or SAML artifacts) for the purpose of establishing a session based on an assertion.
- **Sub-system:** it indicates one of the two parts of Dream project, which are the Forum and the Data aggregator.

### 1.3.2 Acronyms

- **API:** Application Programming Interface
- **HTTPS:** Hyper Text Transfer Protocol over SSL
- **DD:** Design Document
- **ER:** Entity-Relationship
- **TLS:** Transport Layer Security
- **SSL:** Secure Socket Layer
- **DBMS:** DataBase Management System
- **IdP:** Identity Provider
- **RASD:** Requirements Analysis and Specification Document
- **ACS:** Assertion Consumer Service
- **SAML:** Security Assertion Markup Language

### 1.3.3 Abbreviations

- **Gn:** Goal number n
- **Rn:** Requirement number n

## 1.4 Revision history

- v.1.0 - 09/01/2022 - Initial version

## 1.5 Reference documents

- Requirements Analysis Specification Document (RASD)
- Specification document: "R&DD Assignment A.Y. 2021-2022"
- Official Data4Policy stakeholder's project repository https://github.com/UNDP-india/Data4Policy
- Shibboleth documentation https://shibboleth.atlassian.net/wiki/home
- Unified Modeling Language (UML) official specification: https://www.omg.org/spec/UML/
- Archimate official specification: https://pubs.opengroup.org/architecture/archimate3-doc/

## 1.6 Document structure

- **Section 1: Introduction**
  This section offers a brief description of the problem and required functionalities, also providing definition and acronyms that can be found in this document.
  It also provides the revision history and the main structure of the document itself.
- **Section 2: Architectural Design**
  This section is addressed to the developer offering a detailed description of the architecture and its components. The first part describes the chosen paradigm and the division of the system in its layers. Then a better description of modules is given including the general flow for each main function that the system provides.
- **Section 3: User Interface Design**
  This section contains several mockups of the user interfaces and refers to the client side experience. Mockups are provided by means of diagrams in order to describe the general application flow.
- **Section 4: Requirements Traceability**
  This section acts as a bridge between the RASD and DD document, providing a complete mapping of the requirements and goals described in the RASD to the logical modules presented in this document.
- **Section 5: Implementation, Integration and Test Plan**
  The last section describes the procedures for the implementation phase followed by testing and integration. It provides a detailed description of the core functionalities with a complete report about how to implement and test them.

# 2 Architectural Design

## 2.1 Overview

In Figure 1 it's present the black-box view of our architecture, that is defined by three different layers represented:

- **Presentation Layer:** it's the layer that is used to present data to the application layer in an accurate, well-defined and standardized format.
- **Application Layer:** it's the layer that manages all the functions that controls the business logic of our system.
- **Data Layer:** it controls how the data are stored and accessed.

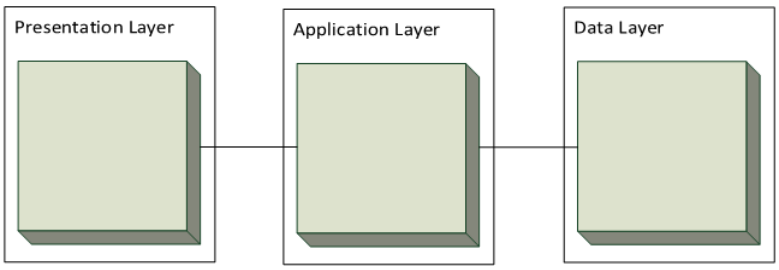


**Figure 1:** Three layers application

A more detailed view of the selected architecture is present in Figure 2:

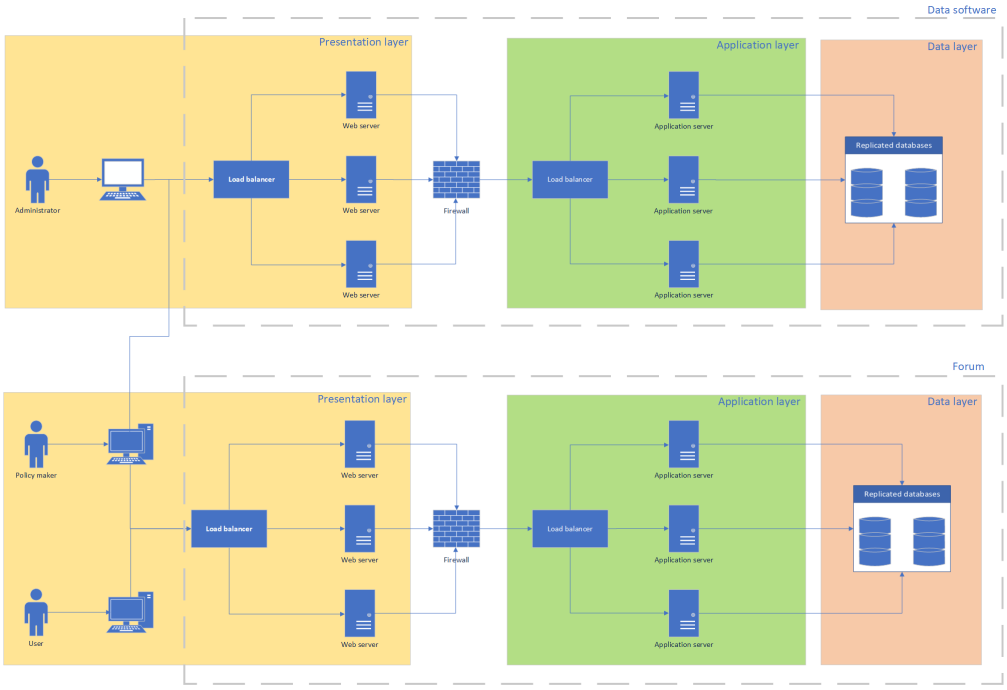


**Figure 2:** Architecture

The service is supposed to be web responsive and for this reason it should be accessible from the vast majority of device from the users. A client-side scripting paradigm will be adopted.

Generally, the architecture divide the application in three different layers: the user can interface with the browser which is connected to replicated Web Servers that will act as middleware, while the last layer is represented by the application servers that contain all the application logic. The DBMS APIs will provide the function to retrieve and store the data by the application servers.

The nodes are separated by firewalls to guarantee a higher level of security of the whole system.

The forum and the data aggregator are two different applications and for this reason they will have two distinct back-end and will use different databases. While Administrator can access only to the data aggregator context and the User can enter only in the forum, the Policy makers have access to both area.
In the following section will be described more in depth the components present in the system.

## 2.2 Component View

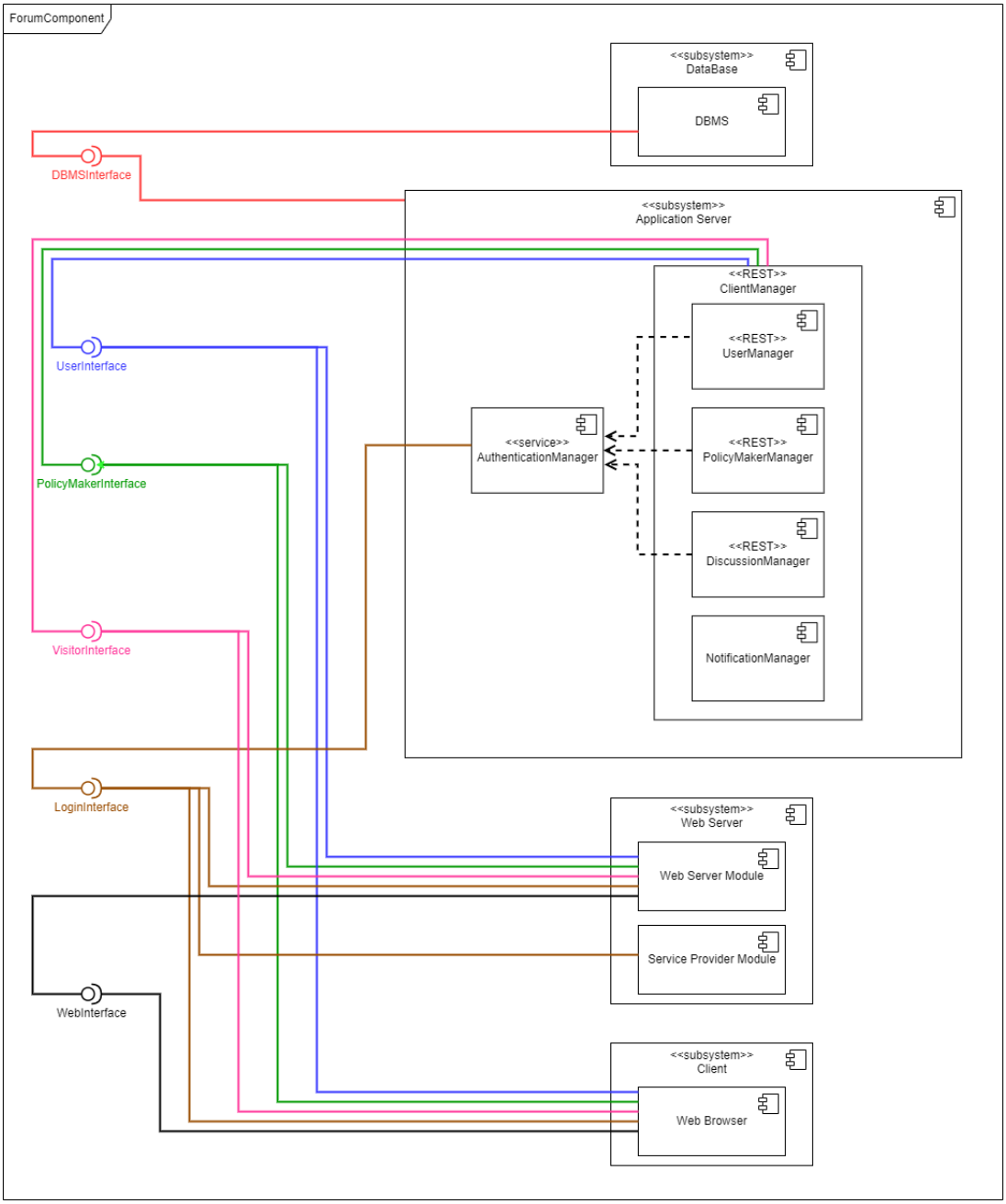


**Figure 3:** Forum Component Diagram

In figure 3 is present a complete diagram representing the layers described before, regarding the forum application. The web server contains two modules:

- **The Web Server Module** is responsible for the web browser request routing, by sending them to the application server, receiving them and sending back responses.
- **The Service Provider Module** generates signed authentication requests to the IdP and receives signed assertion back in order to authenticate the user in the web server session.

Instead, the application server contains different modules:

- **AuthenticationManager**
  It is responsible for managing user authentication and permissions. It is responsible for authenticating the user after reading the session attributes generated by the service provider and then for creating the logged user session (through the LoginInterface). Then, it is also responsible for filtering each incoming request in order to determine if a user has or not the permission for the requested resource. Finally, it is also in charge of detecting if an unauthenticated user is trying to access a resource redirecting it to the service provider module.
- **ClientManager**
  This module manages the requests made by the client. When the user is logged in it provides a UserInterface (if the authenticated person is a User) or a PolicyMakerInterface (if the authenticated person is a Policy maker), exposing functions to manage the entity related information (through the UserManager or the PolicyMakerManager) and the forum content (DiscussionManager).
  In addition the VisitorInterface provides free access to all Visitor in order to get public content.
- **UserManager**
  This module provides the functions related to a User, for instances the possibility to retrieve his own replies or his own information.
- **PolicyMakerManager**
  This module provides the functions related to a Policy maker, for instance the possibility to retrieve a list of Posts or Discussions, in addition to all the functionalities provided by the UserManager.
- **DiscussionManager**
  This module provides all the functionalities related to the forum life-cycle (create, modify and delete content), the authorization is managed by the AuthenticationManager Module.
- **NotificationManager**
  This module handles the email notifications, such as for the publication of a new post or the notification of an approved post.

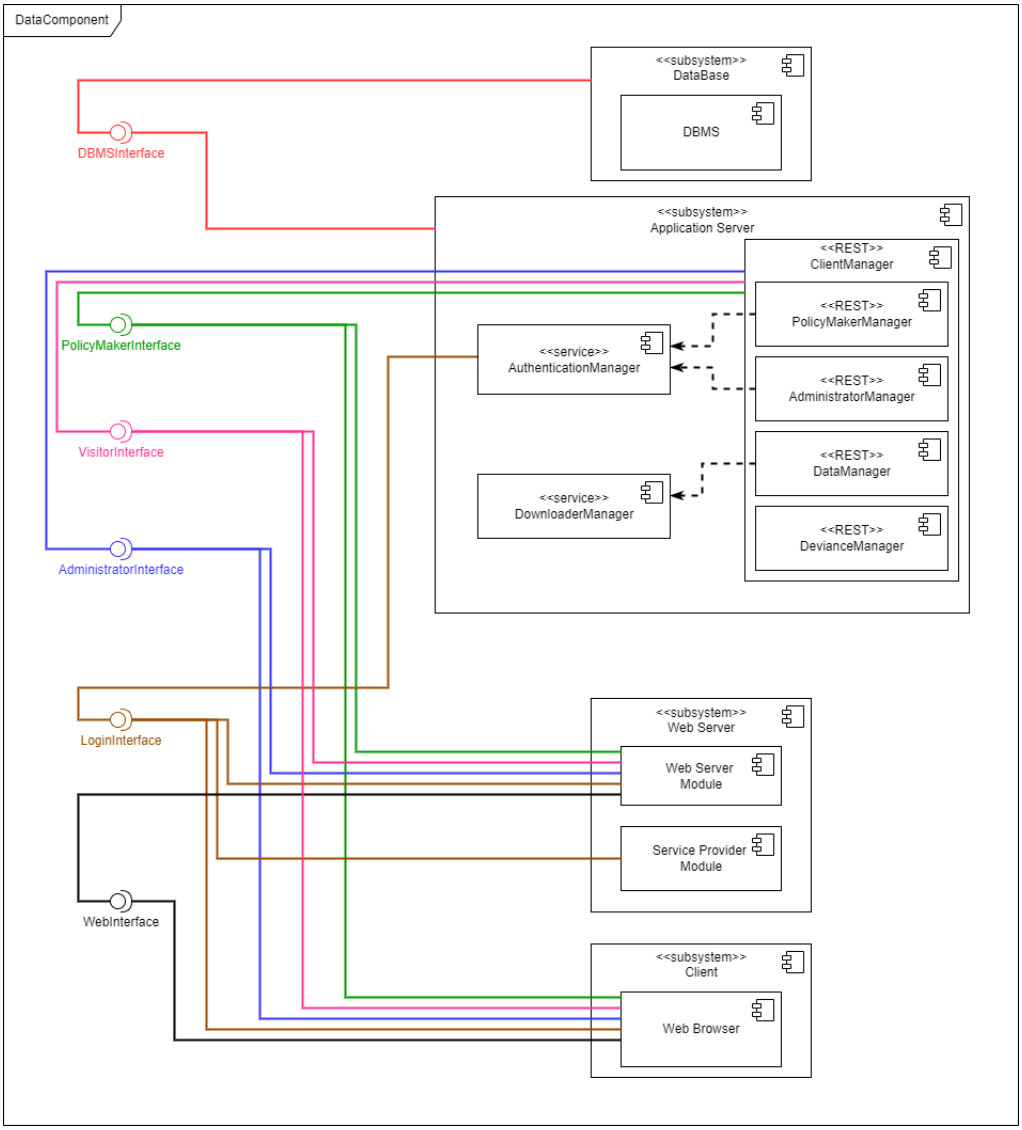


**Figure 4:** Data Component Diagram

In figure 4 is present a complete diagram representing the layers described before regarding data aggregator application.
The web server contains two modules:

- **The Web Server Module** is responsible for web browser request routing sending them to the application server, receiving and sending back responses.
- **The Service Provider Module** generates signed authentication requests to the IdP and it receives signed assertion back in order to authenticate the user in the web server session.

The application server contains different modules:

- **AuthenticationManager**
  It is responsible for managing user authentication and permissions. It is in charge of authenticating the user after reading the session attributes generated by the service provider and then to create the logged user session. Then, it is also responsible for filtering each incoming request to determine if a user whether or not has the permission to the requested resource. It is also responsible for detecting if an unauthenticated user is trying to access a resource redirecting it to the Service Provider Module. Finally, it manages Administrators login exposing the LoginInterface.

- **ClientManager**
  This module manages the requests made by the client. The Administrator-Interface is accessed by Administrators and provides functionalities to add, remove and manage data sources (DataManager and DownloaderManager) while the PolicyMakerInterface provides functionalities to get, analyze and recalculate deviance (DataManager and DevianceManager). Both the type of authenticated people have their own Manager (AdministratorManager and PolicyMakerManager) that provides functionalities related to the entity (e.g. personal information).

- **PolicyMakerManager**
  This module provides the functions related to a Policy maker, for instance the possibility to retrieve the area he is assigned to and his own information.

- **AdministratorManager**
  This module provides the functions related to an Administrator, for instance the possibility to add and remove other Administrators.

- **DataManager**
  This module provides all the functionalities related to the data management such as add and remove a data source but also to obtain and filter present data.

- **DownloadManager**
  This module is responsible for connecting and fetching periodically new data from public data sources, filter and store them.

- **DevianceManager**
  This module provides all the functionalities to calculate a new deviance and get an existing one. This module is used both by a Policy maker and by the system itself for periodic computation.

## 2.3 Deployment View

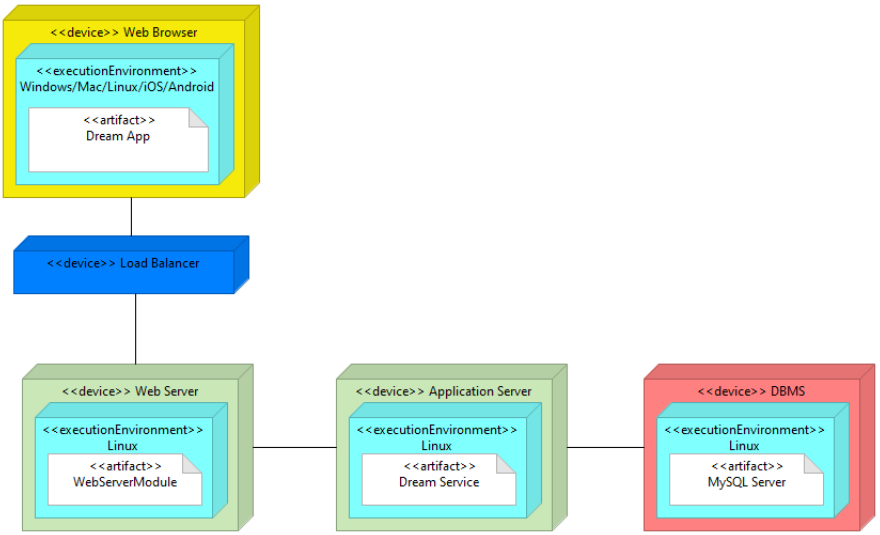


**Figure 5:** Deployment Diagram

The deployment diagram in figure 5 shows the topology of the system's hardware and specify the distribution of components. For each device its Operating System is indicated.

- **Tier 1**: It is the client machine. It could be any device provided with a Web Browser running on Windows, Mac or Linux for desktop devices or in iOS or Android for mobile devices.
- **Tier 2**: It consists of the load balancer and the replicated web servers. The first one is a device that allows to balance the workload between servers, to maintain their capacity at an optimal level. This enables a server cluster to handle peak traffic, and provides a backup solution in the event of an outage. Replicated web servers display website content through storing, processing and delivering web pages to users. They do not execute any business logic but only respond to client requests made over the World Wide Web. They also contains the styling logic of the page.
- **Tier 3**: it consists of the application servers. They provides all the business logic, allowing to communicate to the client tier using APIs and connect to the data tier using the DBMS gateway.
- **Tier 4**: it consists of the database management system servers. The data are stored in those replicated devices and they provide to the application server tier the operations needed to manage the data.

## 2.4 Runtime View

### 2.4.1 Visitor filters data

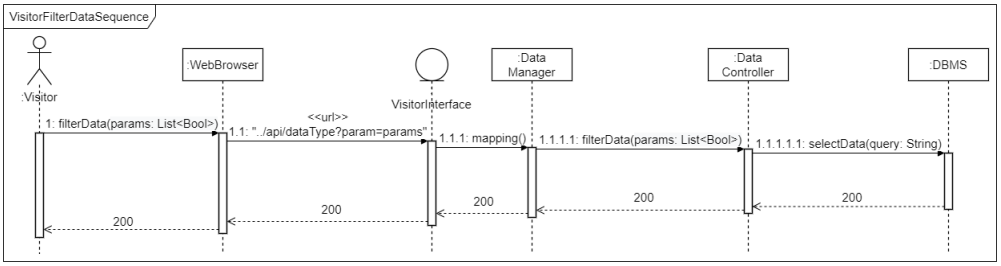


**Figure 6:** Visitor filters data sequence diagram

An unregistered Visitor can accesses to the data provided by the platform. In order to consult them more clearly, the visitor can select different filters that allow viewing only the data of interest.
Once he has selected the parameters of interest he clicks on the "Filter" button; at this point the Web Browser sends the data to the DataManager through the VisitorInterface ("../api/dataType?param=params"). The DataManager interfaces with the DataController which queries the DBMS.
Based on the returned value, the DataController sends a 200 response's status code and the Web Browser displays the required data set.

### 2.4.2 Visitor downloads data

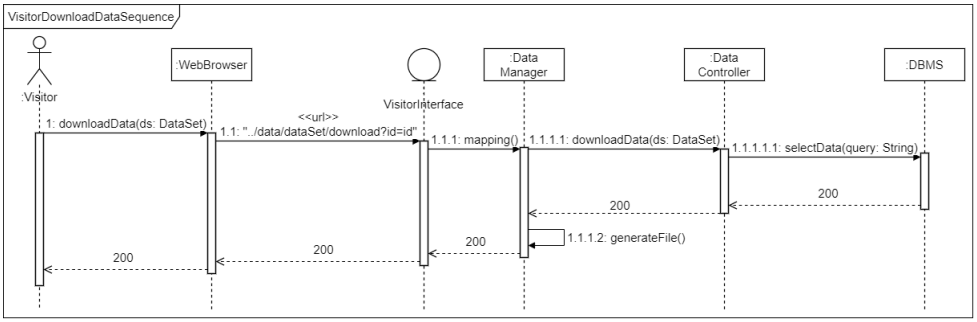


**Figure 7:** Visitor downloads data sequence diagram

Everyone using the platform can also download data sets of interest.
Clicking the "Download" button associated to the data set of interest the Web Browser will send the request to the DataManager through the VisitorInterface ("../data/dataSet/download?id=id"). The DataManager calls the DataController to query the DBMS asking for the data set indicated. Once the DataManager is provided with the required data, it generates a file in the selected format and the file is returned to the Visitor with a successful response status code.

### 2.4.3 Visitor enters a discussion

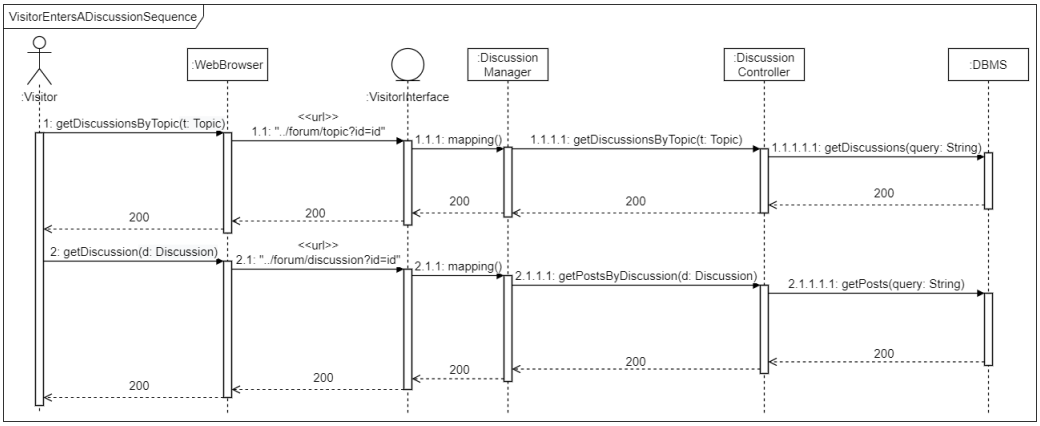


**Figure 8:** Visitor enters a discussion sequence diagram

In order to consult the forum discussions everyone is able to navigate through the different topics.
To do so the first request is send by the Web Browser to the DiscussionManager via VisitorInterface ("../forum/topic?id=id"). The DiscussionManager will retrieve the list of discussions related to the topic selected from the DBMS using the DiscussionController. This list will be returned to the Visitor and he will be able to select the one he wants to consult. The Web Browser will forward the request to the DiscussionManager which will call the DiscussionController in order to retrieve from the DBMS the post related to that specific discussion.
At this point a post list will be returned to the Visitor and a 200 response status code will confirm operation success.

### 2.4.4 Sign Up

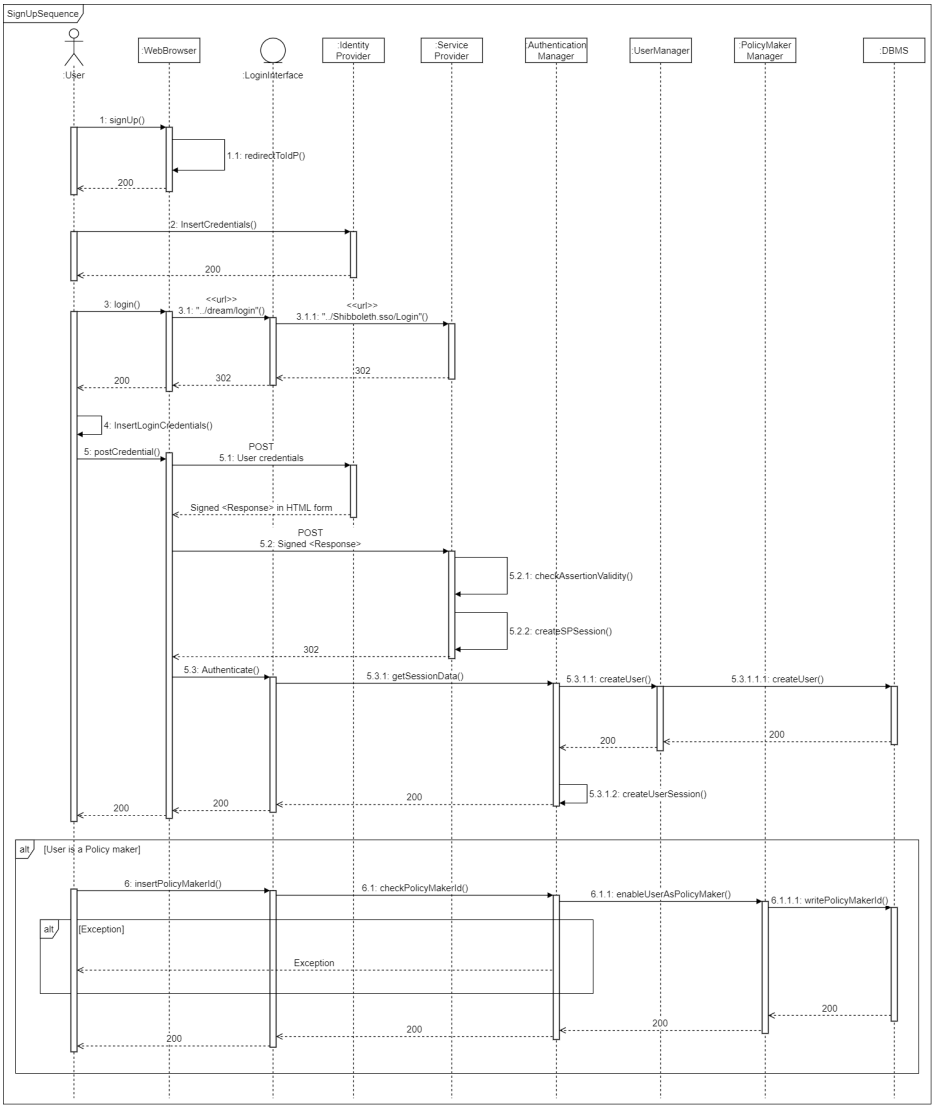


**Figure 9:** Sign Up sequence diagram

In order to Sign Up a User is redirect by the Web Browser to the default Identity Provider where he can register his account. When the account creation is completed the User can start again by login. The login action performed by the user is translated in a url ("../dream/login") which is redirected to the Service Provider login initiator url ("../Shibboleth.sso/Login") by the LoginInterface. Now the user is redirected in the Identity Provider login page with a signed authentication request, from where he can login with the account created before. If the IdP authentication is completed successfully a POST to the Service Provider ACS ("../Shibboleth.sso/SAML2/POST")

is performed automatically with a signed assertion (authentication response). The Service Provider checks the validity of the received assertion and saves the user data in the Session and returns a redirect request to the dream single-sign-on page managed by the AuthenticationManager. Now the session data are read, a new user is created by the UserManager and saved in the database by the DBMS. Once the new user is created the AuthenticationManager creates a local user session to complete the login action. If the received user role is compatible with the Policy maker one, the user is asked to insert a PolicyMakerID to verify the new account, the inserted value is sent to the AuthenticationManager to be verified and, if correct, the PolicyMakerManager proceeds to update the user in the DBMS with the new role.

### 2.4.5 Login

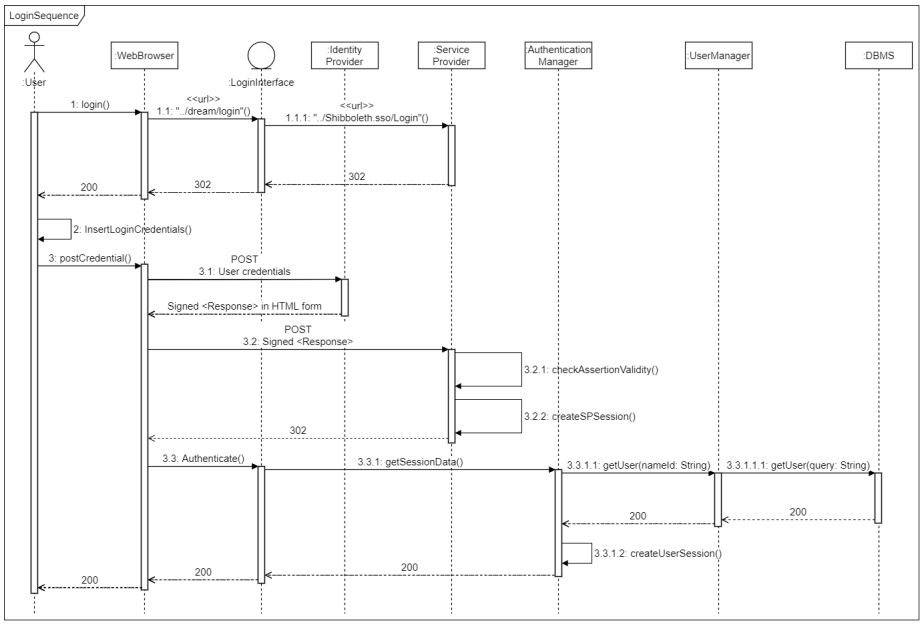


**Figure 10:** Login sequence diagram

In order to login a User or a Policy Maker the web browser sends a request to the loginInterface ("../dream/login") which will invoke the Service Provider. The provider will return a temporary redirection (http code:302) to the Identity Provider containing a signed authentication request. The Identity Provider will recognize the service request and let the user to authenticate with the IdP credential. After the user is logged in the IdP, it will post the signed assertion (authentication response) to the Service Provider ACS with the user requested data that will be saved in the server session. This will let the AuthenticationManager to create a session retrieving the data from the DBMS passing through the UserManager's getUser() request. At the end a 200 response status code will be sent to the User.

### 2.4.6 Login Administrator

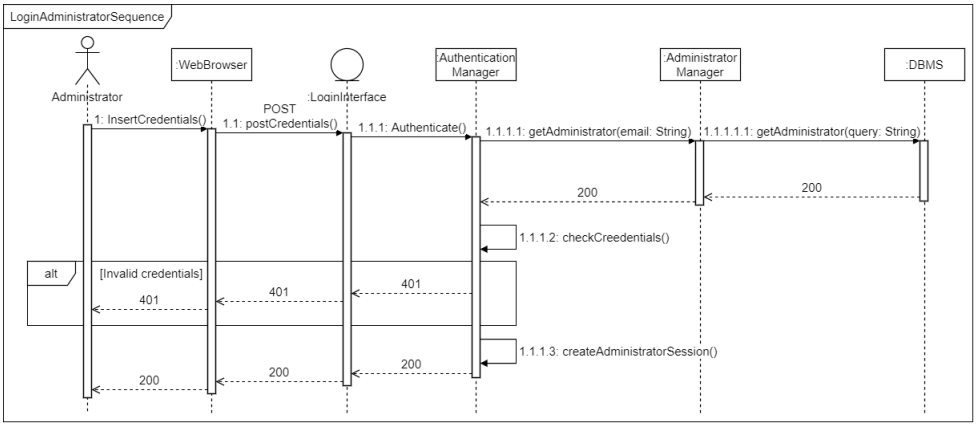


**Figure 11:** Login Administrator sequence diagram

The Administrator login process works differently because it doesn't relies on an external services.
First of all, the Administrator will insert the login credentials and those will be sent to the LoginInterface. The AuthenticationManager invoked by the interface will retrieve the Administrator data from the DBMS calling the AdministratorManager. AuthenticationManager at this point checks the credentials: if they are invalid it will respond with a 401 (unauthorized) http status code, otherwise an Administrator session will be created and a 200 response status code will be sent to the Administrator.

### 2.4.7 Publish a post by User

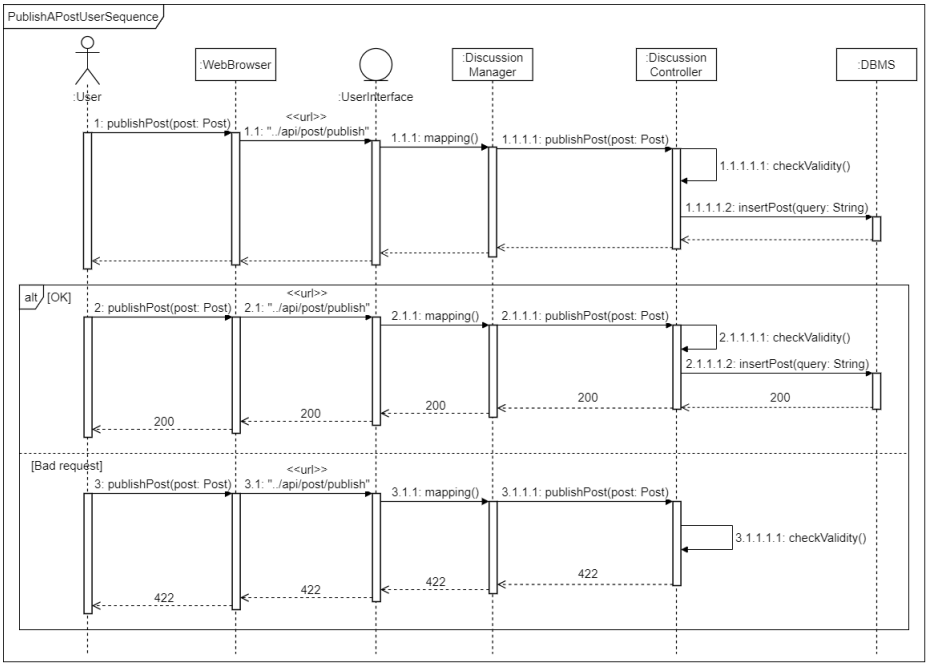


**Figure 12:** Publish a post by User sequence diagram

After writing the post, the User tries to publish it by clicking on publish a post in the Web Browser. This will contact the DiscussionManager via UserInterface ("../api/post/publish").
The DiscussionManager then calls the DiscussionController which sends an insert query to the DBMS. The post will be added to the post table with pending status, waiting for Policy maker approval.

### 2.4.8 Publish a post by Policy Maker

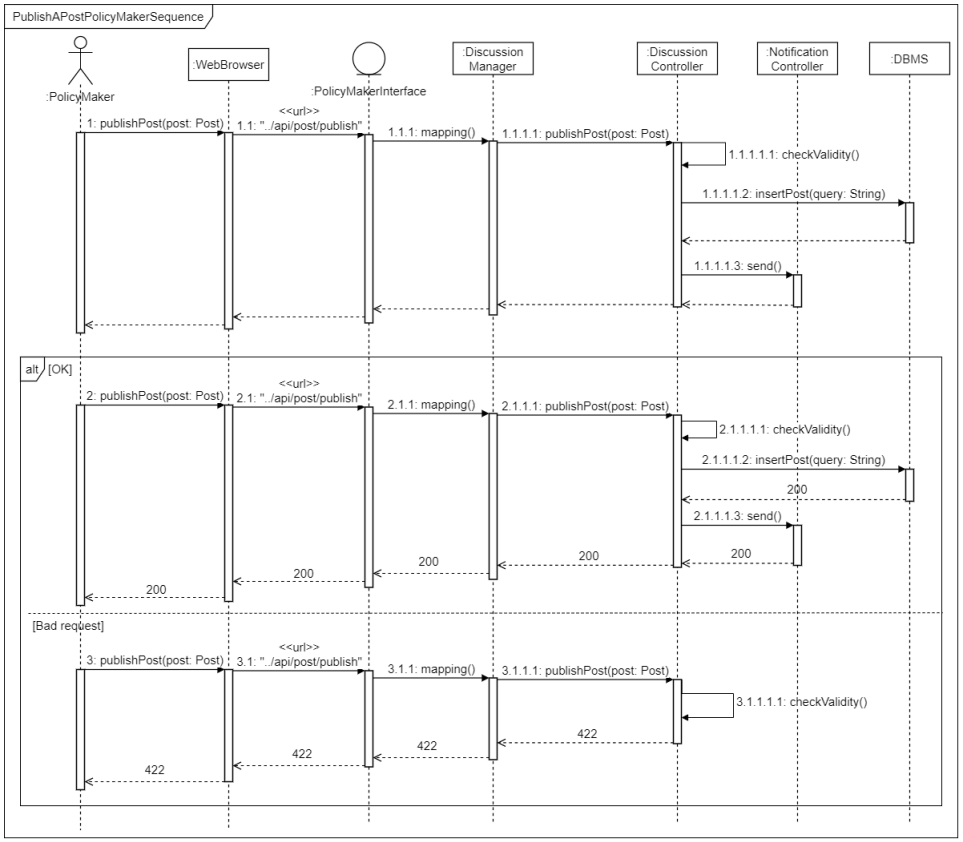


**Figure 13:** Publish a post by Policy Maker sequence diagram

It is a very similar process to the one ran by the User but it has some relevant differences.
When a Policy maker tries to publish a post, the PolicyMakerInterface, invoked by the Web Browser("../api/post/publish"), contacts the DiscussionManager which inserts the post in the DBMS via the DiscussionController. In this case the post is immediately available from the forum and doesn't need for approval. The DiscussionController will then invoke the NotificationController in order to send a notification to the list of users that follow the discussion.

### 2.4.9 Modify a post

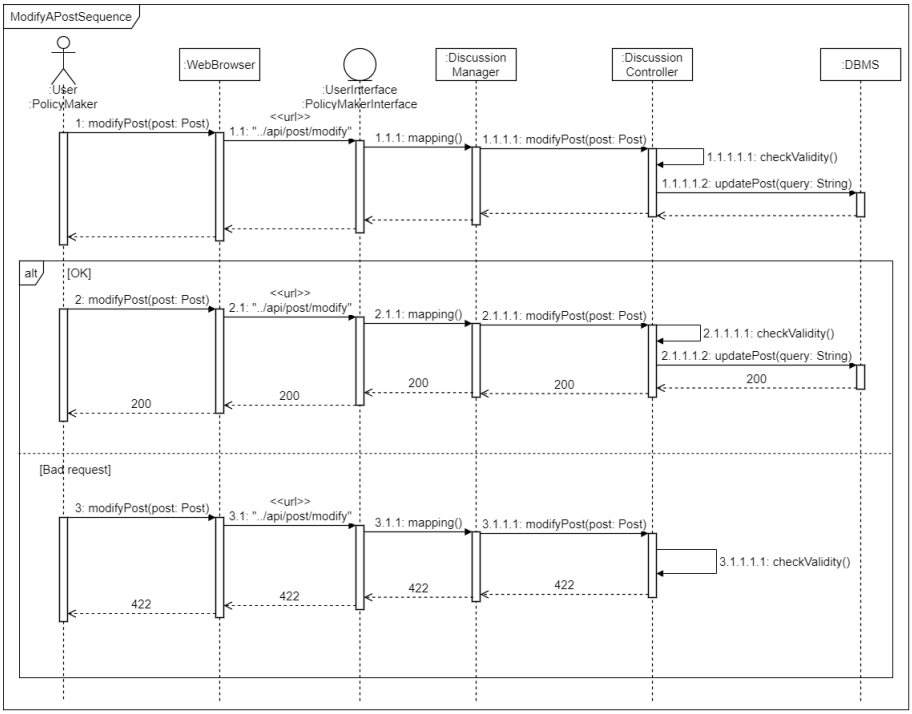


**Figure 14:** Modify a post sequence diagram

When a User or a Policy maker wants to modify a post, after making the desired changes, the DiscussionManager is invoked by the Web Browser via the UserInterface or the PolicyMakerInterface("../api/post/modify").
The DiscussionManager then calls the DiscussionController which queries for an update in the indicated post. If the operation is successful, a 200 response status code is returned to the User/Policy maker.

### 2.4.10 Delete a post

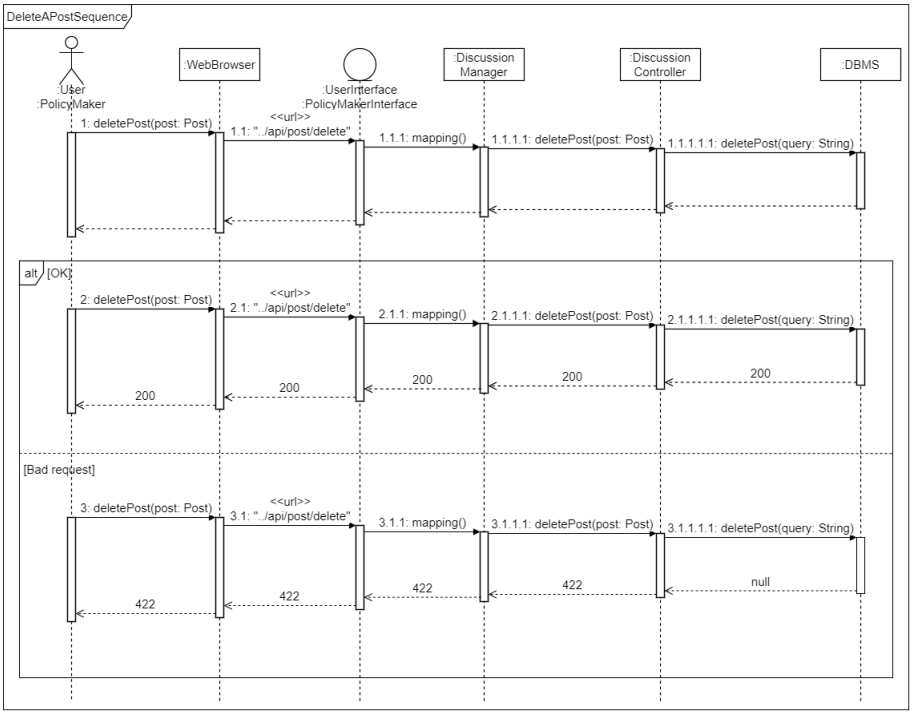


**Figure 15:** Delete a post sequence diagram

When a User or a Policy maker wants to delete a post, the DiscussionManager is invoked by the Web Browser via the UserInterface or the PolicyMakerInterface("../api/post/delete").
The DiscussionManager then calls the DiscussionController which queries for a delete of the post. If the operation is successful, a 200 response status code is returned to the User/Policy maker.

### 2.4.11 Create a discussion

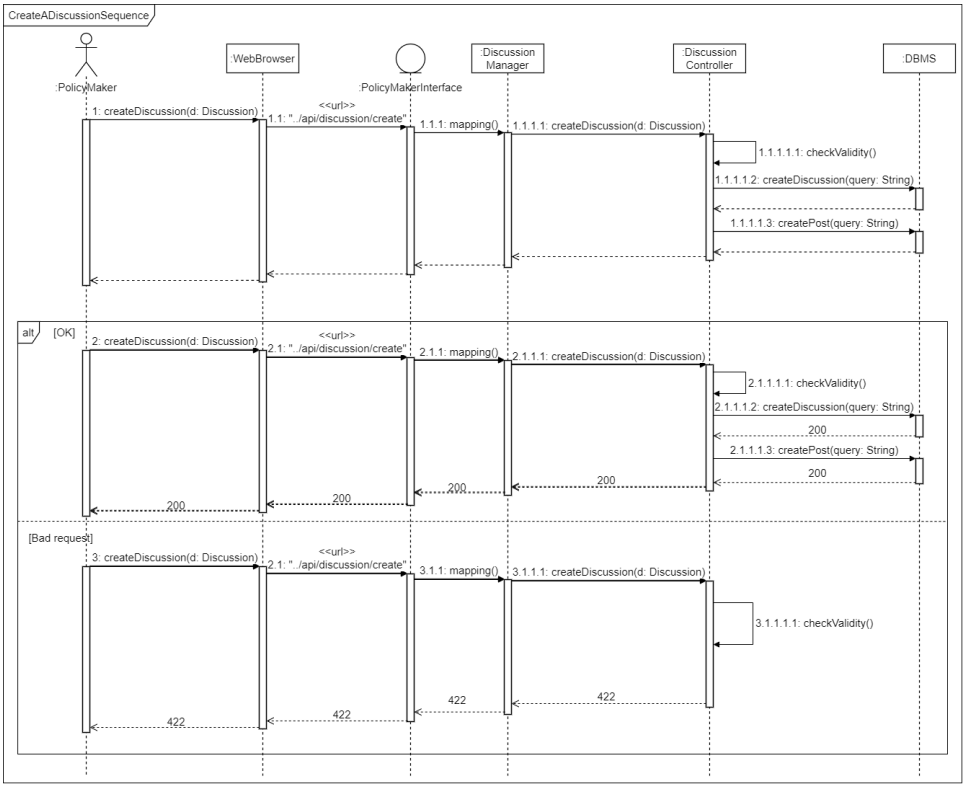


**Figure 16:** Create a discussion sequence diagram

A Policy maker, in order to create a discussion, will use the "create a discussion" button that will call the PolicyMakerInterface ("../api/discussion/create"). The request will be forwarded to the DiscussionManager, which will query the DBMS via the DiscussionController in order to insert a new discussion. Every discussion requires at least one post, so when the DBMS will insert the new discussion the DiscussionController will ask the DBMS to insert the post passed into the Discussion object. A 200 response status code will inform the Policy maker that the operation was successful.

### 2.4.12 Delete a discussion

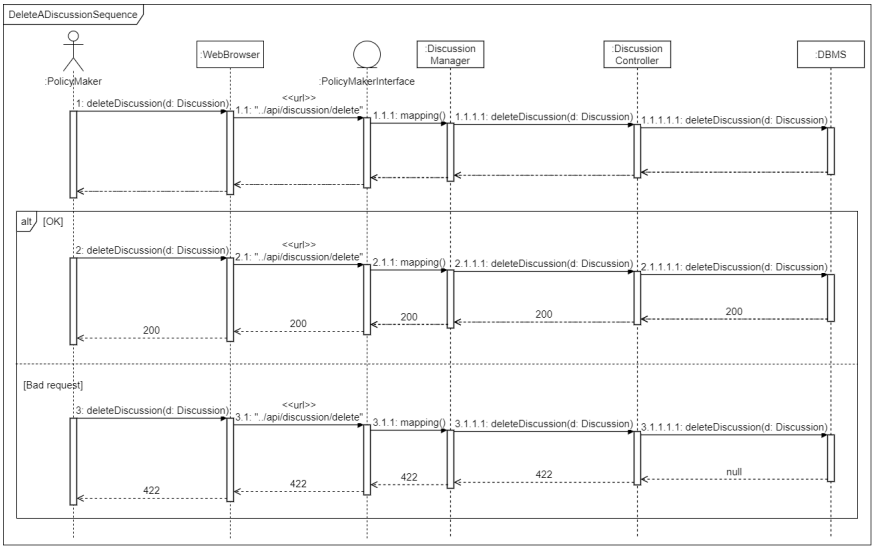


**Figure 17:** Delete a discussion sequence diagram

In order to remove a whole discussion from the forum, a Policy maker will invoke the PolicyMakerInterface via Web Browser("../api/discussion/delete"). The request will be forwarded to the DiscussionManager, which will call the DiscussionController in order to remove the discussion and the relative posts from the DBMS. A 200 response status code will inform the Policy Maker that the operation was successful.

### 2.4.13 Confirm pending post

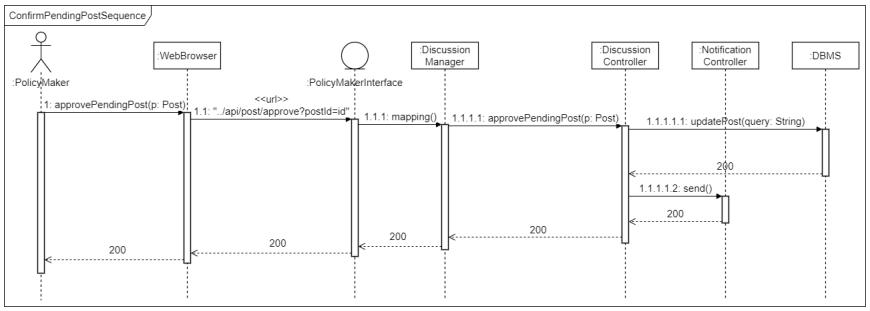


**Figure 18:** Confirm a pending post sequence diagram

Before a post written by a User is visible in the forum, it is necessary that the Policy maker approves its publication.
To do so, a Policy maker will click on "Approve pending post" button and the Web

Browser will call the PolicyMakerInterface( "../api/post/approve?postId=id"). The request will be forwarded to the DiscussionManager and the DiscussionController will be invoked to update the post's status field. Then, the DiscussionController will call the NotificationController in order to notify its publication in the forum to the creator of the post and to the other users following the discussion in which that post is being published. A 200 response status code will inform the Policy maker that the operation was successful.

### 2.4.14 Decline pending post

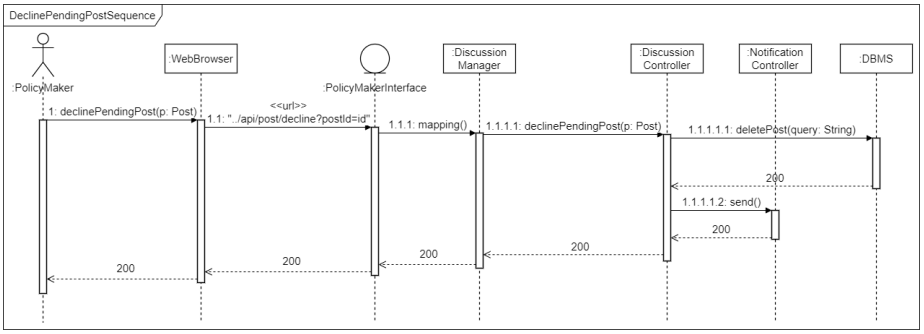


**Figure 19:** Decline a pending post sequence diagram

Some posts should not be made public because they do not follow the forum rules or are not related to the discussion topics.
To do so, a Policy maker will click on "Decline pending post" button and the Web Browser will call the PolicyMakerInterface( "../api/post/decline?postId=id"). The request will be forwarded to the DiscussionManager and the DiscussionController will be deleted from the DataBase. Then, the DiscussionController will call the NotificationController in order to warn the creator that his post will not be published in the forum. A 200 response status code will inform the Policy Maker that the operation was successful.

### 2.4.15 Recalculate new Deviance

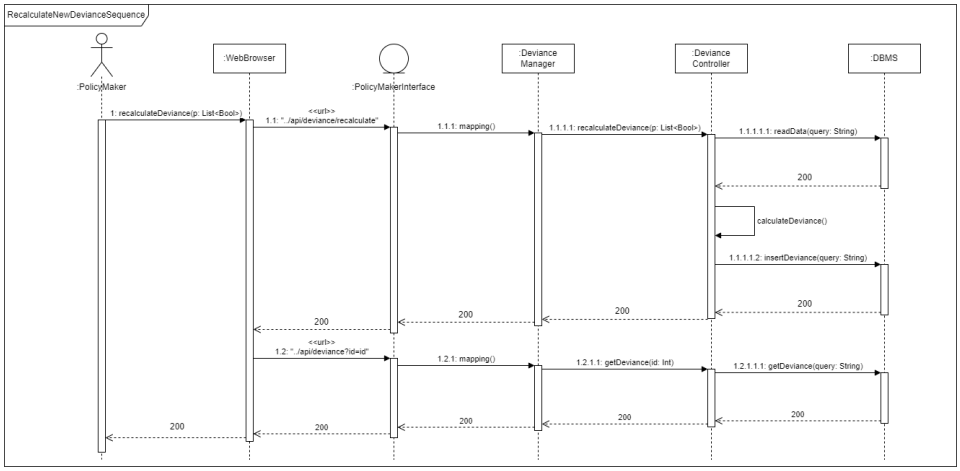


**Figure 20:** Recalculate new Deviance sequence diagram

A Policy maker might want to recalculate the Deviance using a different set of parameters than the standard one.
To do this, once he has selected all the parameters of his interest, the Policy maker will call the "recalculateDeviance" function and the Web Browser will forward his request to the PolicyMakerInterface ("../api/deviance/recalculate"). This request is submitted to the DevianceManager which queries the DBMS via the DevianceController. Using the data retrieved the DevianceController will be able to calculate the new results and will insert them into the DataBase.
When a 200 response code status arrives to the Web Browser, it will automatically sent a request to refresh the view in order to display the new results.
The PolicyMakerInterface is invoked by the Web Browser ("../api/deviance?id=id"), which submits the request to the DevianceManager. The DevianceManager will retrieve the new data from the DBMS via the DevianceController. A 200 response code status delivered to the Policy maker will tell him that the operation was successful.

### 2.4.16 Add a new data source

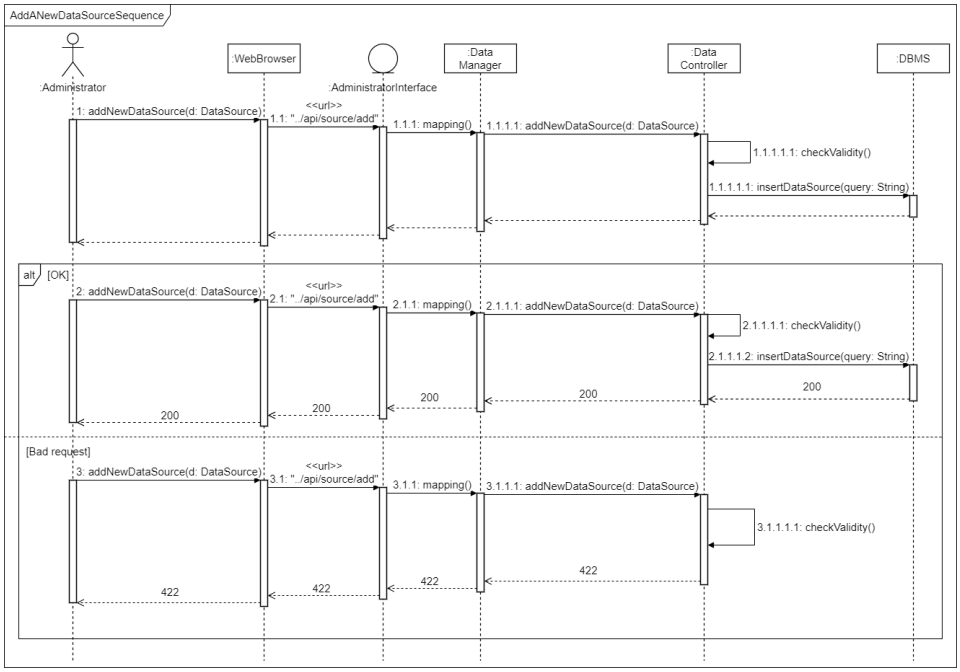


**Figure 21:** Add a new data source sequence diagram

To add a new data source, an Administrator will have to invoke the "addNew-DataSource()" method.
This will call the AdministratorInterface via Web Browser ("../api/source/add"). The interface calls the DataManager which will insert the data source in the DBMS via DataController. A 200 response code status delivered to the Administrator will let him know that the operation was successful, otherwise a 422 response status code will tell that the request wasn't able to be processed.

### 2.4.17 Modify a data source

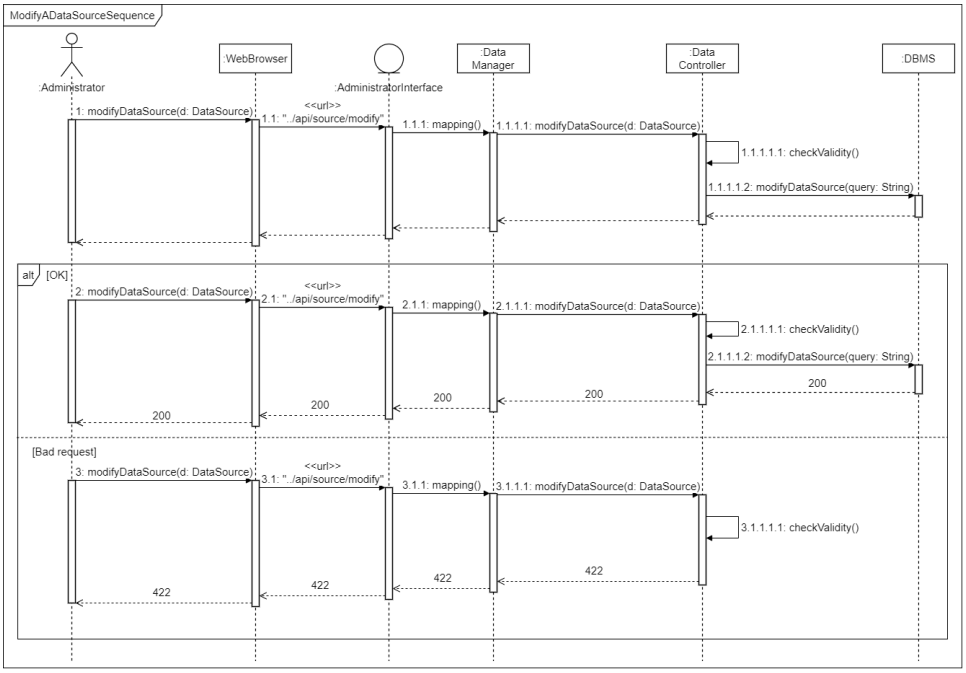


**Figure 22:** Modify a data source sequence diagram

In order to modify a data source an Administrator invoke the AdministratorInterface via Web Browser ("../api/source/modify"). The data are sent to the DataManager which will update the DBMS record calling the DataController. A 200 response code status delivered to the Administrator will let him know that the operation was successful, otherwise a 422 response status code will tell that the request wasn't able to be processed.

### 2.4.18 Remove a data source

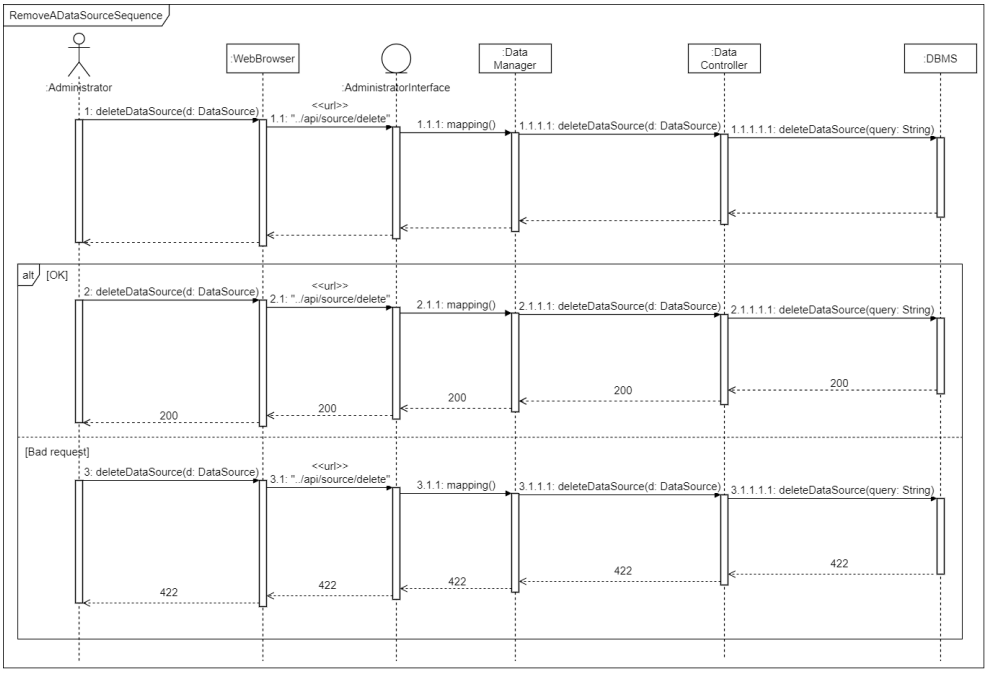


**Figure 23:** Remove a data source sequence diagram

An obsolete data source can be removed by an Administrator.
To do so the AdministratorInterface is invoked via Web Browser ("../api/source/delete"). The interface calls the DataManager which will remove the data source from the DBMS via DataController. A 200 response code status delivered to the Administrator will let him know that the operation was successful, otherwise a 422 response status code will tell that the request wasn't able to be processed.

## 2.5 Component Interfaces

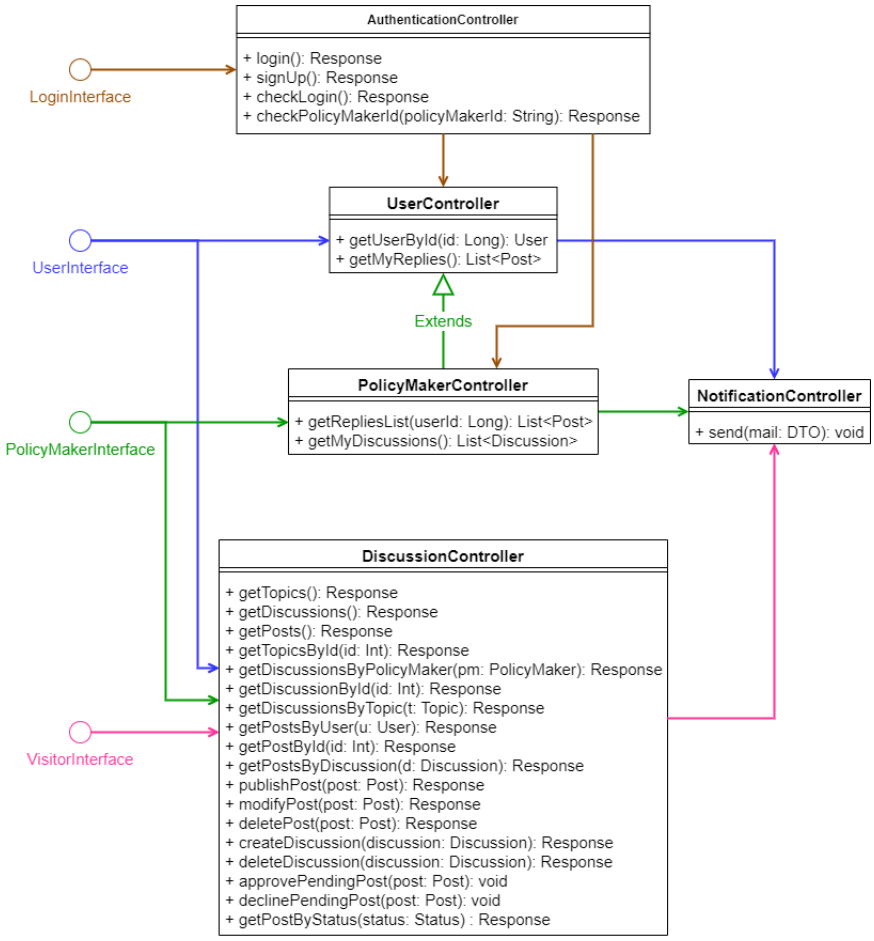


**Figure 24:** Forum Interfaces Diagram

In figure 24 the principal interfaces to make Dream Forum run are shown.
Starting from the beginning the LoginInterface is needed to make the login and the sign up possible. It is realized through the AuthenticationController which is also used by the UserController and the PolicyMakerController to check permissions in the operations.
The UserInterface provides all the functionalities needed by a User such as the management of his account, the possibility to retrieve his own replies and to create, modify and delete a post. These functionalities are implemented by two controllers: the UserController and the DiscussionController.
The PolicyMakersInterface is quite similar to the User one but provides additional methods for the forum moderation such as the creation of a discussion or the possibility to modify/delete other User's posts. These functionalities are implemented in the PolicyMakerController (an extension of the UserController) and the Discus-

sionController.
The NotificationController is used by the UserController, the PolicyMakerController and the DiscussionController to send email notification when needed and, finally, the VisitorInterface relies on the DiscussionController in order to let visitors to access public content.

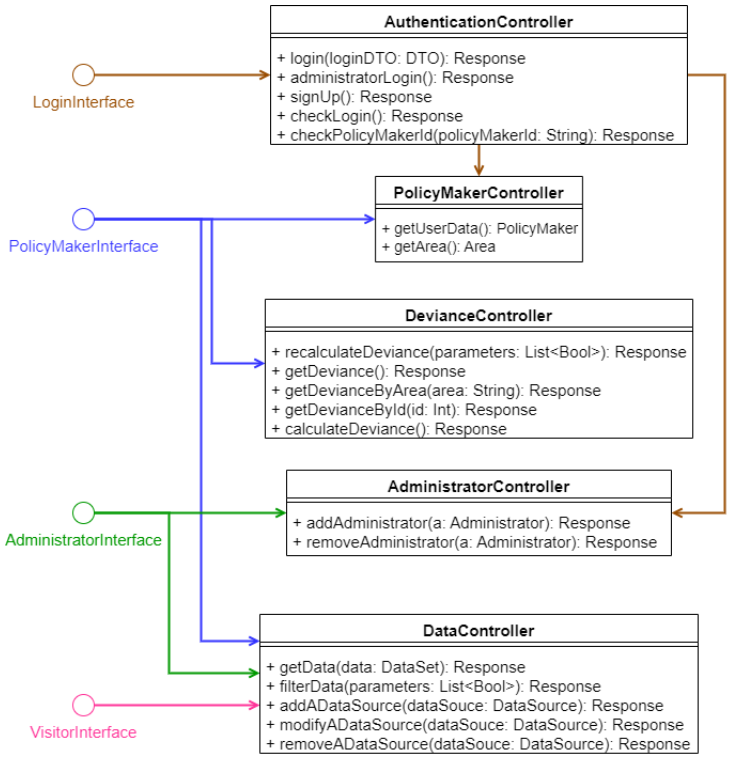


**Figure 25:** Data Interfaces Diagram

In figure 25 the principal interfaces to make Dream Data aggregator run are shown.
Starting from the beginning, the LoginInterface is required to let Administrators (local authentication) and Policy makers (IdP authentication) to login. It uses the AuthenticationController which provides both the function for the Administrators authentication and the external one.
The PolicyMakerInterface provides all the functionalities used by a Policy maker such as getting his own information (handled by the PolicyMakerController) and deviance data that is handled by the DataController.
The AdministratorInterface, instead, provides the admin functionalities like adding or remove a new administrator by means of the AdministratorController and add, modify and remove data-sources with the help of the DataController.
Finally, the VisitorInterface relies on the DataController which let them to retrieve all the public content they can access to.

## 2.6 Logical Description of Data

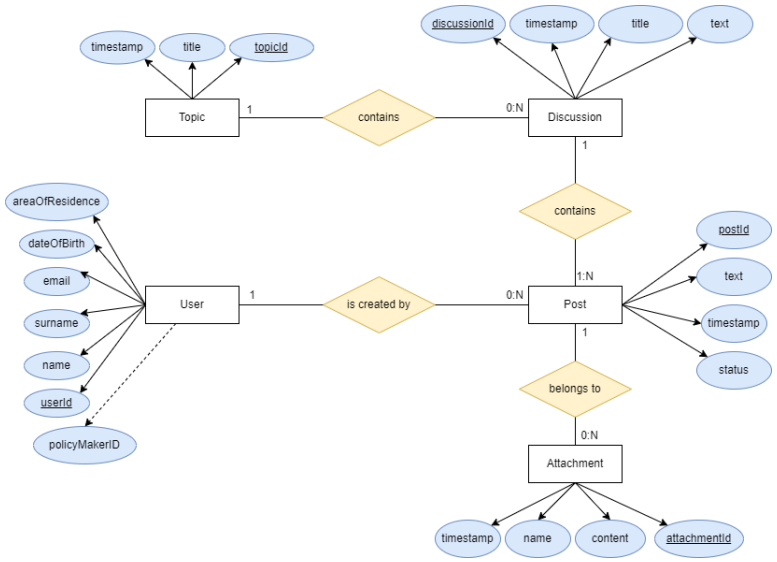


**Figure 26:** Forum ER Diagram

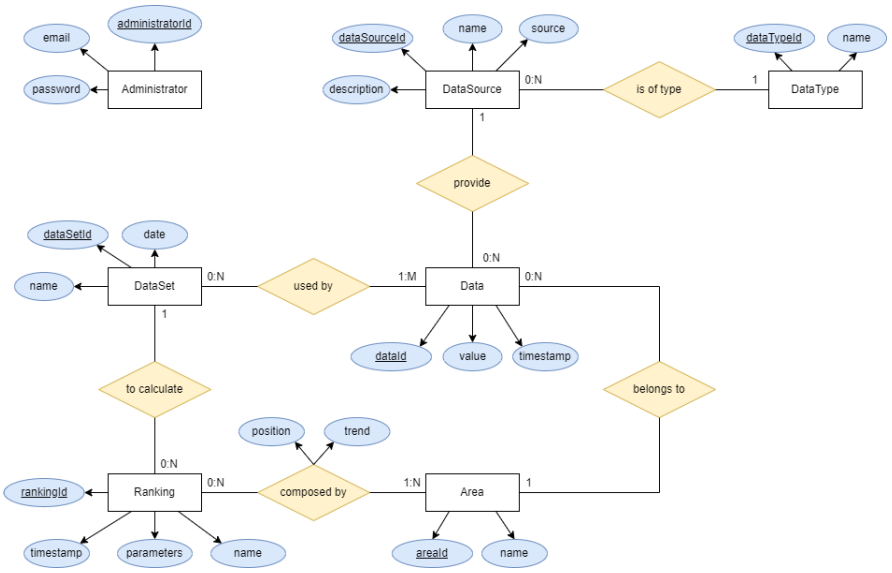


**Figure 27:** Data ER Diagram

The two figures above show the database structure for the forum and the data sections.
In the first one the hierarchical structure of the forum (fig: 26) is evident thanks to the division in topic, discussion and post. The topic category may not contain any discussion in a first moment, but the discussion category needs to always have at

least one post related. A post is connected also to his creator, which can be either a User or a Policy maker. The Policy maker differs from the User because it is identified with a non-null value in the policyMakerID field. The post table is also connected to the attachment one: a post can have more than one attachment.

The Data ER diagram (fig: 27) shows how the different information that concern the data that Dream platform provides needs to be organized. An administrator table will be necessary to store the Administrator login credentials because they don't rely over the external provider. The data source will be linked to the data it provides and in the DataType table will be presented the data type related to the data supplied. The adoption of a table instead of using a simple enumeration consents to dynamically change the data type presents in the database. Data will be linked to the Area of their competence. A Data set will be established by different data and those data could be used by different data set. This is why a bridge table will be needed to link the two tables. Each data set will be used to calculate different ranking using the Deviance algorithm and these ranking will be characterized by a parameter selection.

## 2.7 Architectural Style and Patterns

### 2.7.1 Four-tiered architecture

The usage of a four-tier architecture allows having four layers with completely different goals:

- Presentation Layer (PL);
- Data Presentation Layer (DPL);
- Business Logic Layer (BLL);
- Data Access Layer (DAL).

Each tier is responsible for a specific layer: this implies that the logic is separated and changes at a certain tier, doing like that will not affect the other layers. This improves the maintainability of code and it is easier to implement new features.
Another important aspect is performance: caching in the presentation tier allows to reduce the network usage and the workload on the Application and Data tiers. Thanks to this architecture, it is also possible to optimize the management of requests because of the load balancer action.
This architectural style allows to improve security because client can not have direct access to database and the firewall prevents unauthorized access to the internal network.
Finally, the application should be more scalable and the use of replicated servers allows to have better availability at the same reliability level for each server.

### 2.7.2 RESTful Architecture

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. This lets to reduce the bandwidth used by the application: the server is queried only for retrieve and modified data, while the operations are managed directly on client side.
This behavior guarantees less computational load on the server and a dynamic attitude of the service, making it more suitable for efficient internet usage and prevent refreshing the page each time an action is made, improving the user experience.

### 2.7.3 Model View Controller (MVC)

Model-View-Controller is a software design pattern used for developing User interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information, from the ways information is presented to and accepted from the user.
These three components are:

- **Model:** the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- **View:** any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart in the dashboard page and a tabular view in the Deviance one.
- **Controller:** accepts input from the client and converts it to commands for the model that generally lead to state changes in the view.

## 2.8 Other Design Decision

### 2.8.1 Scale-out

We approach to this design decision because following it, the system will clone the nodes in which it is more probable to appear a bottleneck, in order to increase the overall scalability.
This design decision brings to have a higher deployment effort, but also a lower hardware upgrade cost, when the system has reached its limit. All those considerations leads to the conclusion that the scale-out decision should be adopted by the system.
In the end, the system needs a load balancer to correctly balance the load of the incoming requests to the node with more available computational resource.

### 2.8.2 Easy usability

Since our target is a customer of every age, the system is designed to be very simple and intuitive, so particular attention must be put on:

- Forum: it is designed to be minimal, but at the same time full featured.
- Public data access: the data set are easily accessible through the Home Page of the Dream site and once reached, filtering and downloading through the different data set is really intuitive.

### 2.8.3 Reliability and Availability

Thanks to the Scale-out design decision present above (2.8.1), consisting in different physical nodes working in parallel, the system will prevent data loss overall and will also avoid system downtimes, incrementing the overall workload.
For instance, if one node fails while an User is trying to access the API's, other servers that works in parallel with it will supply the requested service. The scope is to obtain a system with a minimum availability of 99%, regarding the Data Manager component, while the forum component could have lower value, possibly higher then 90%.

### 2.8.4 Security

The system has an encrypted communication between parts going on a secure channel using SSL protocol over HTTP (obtaining HTTPS) and also the system can be integrated with institutional IdP, since the authentication and authorization are performed through SAML2.0 protocol.

### 2.8.5 Modularity and Maintainability

The system is designed to be highly reusable, meaning that it must be composed by different modules and this design decision facilitate further maintenance to the system.

# 3 User Interface Design

In this section is represented the design of the main screens of our site Dream, with a description of the flows of the most relevant functionalities divided by type of user. We have used different mockups already present in the RASD but several mockups have been added in order to better describe every flow.

## 3.1 User interfaces

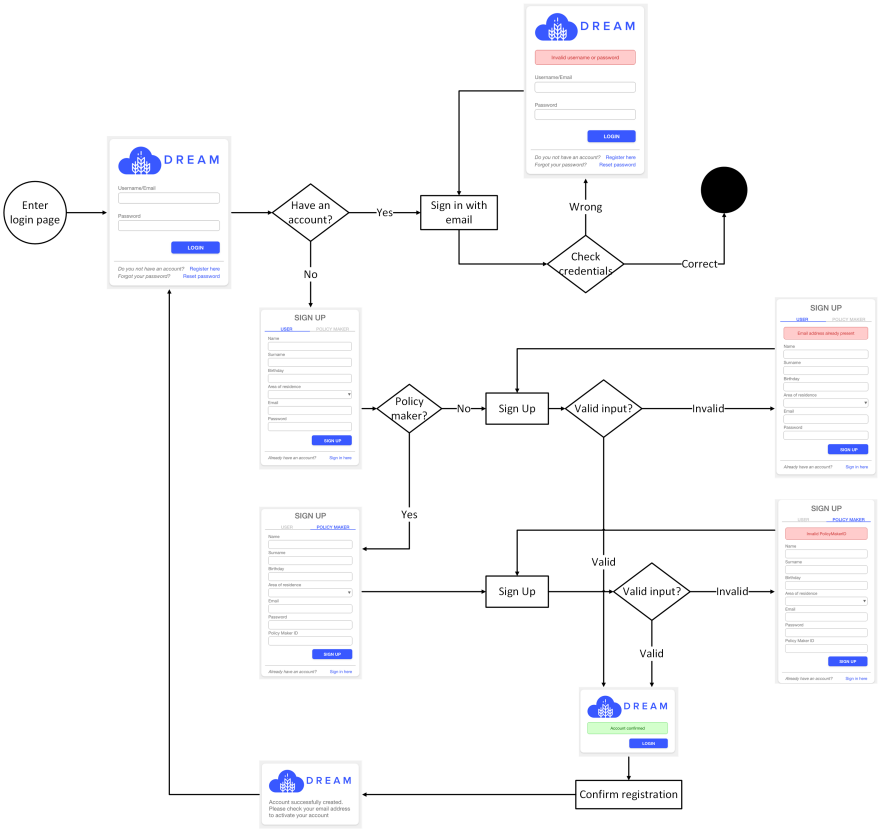


**Figure 28:** User Sign Up and login

## 3.2 Administrator interfaces

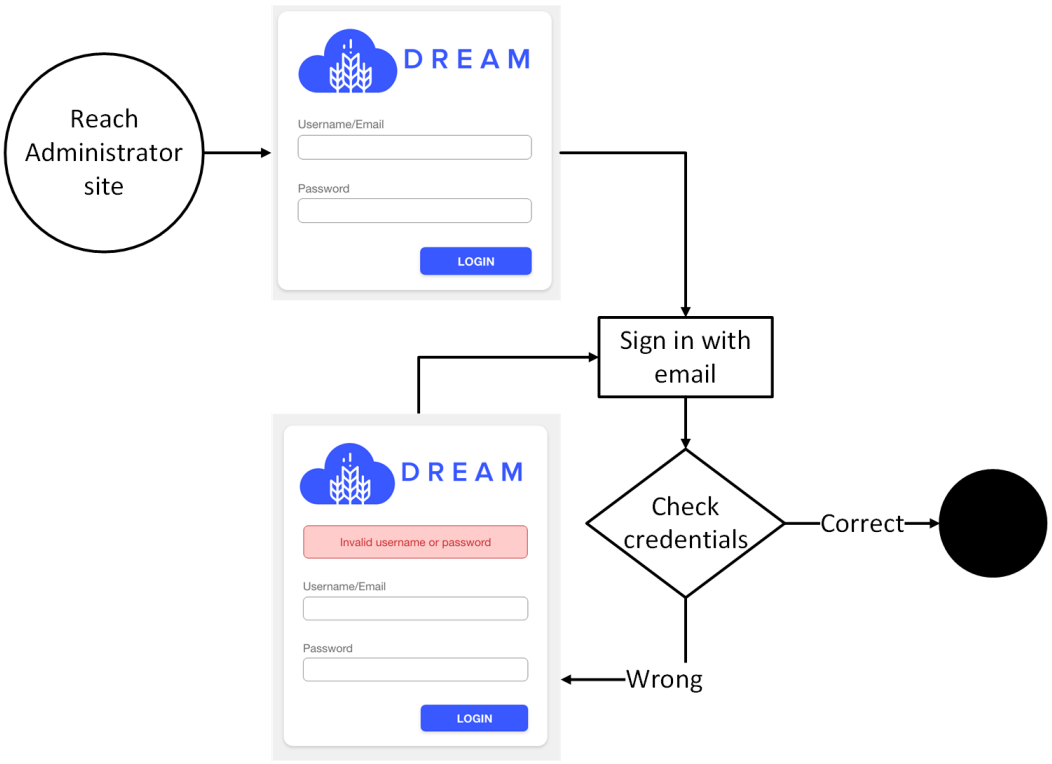


**Figure 29:** Administrator Login

## 3.3 Policy maker interfaces

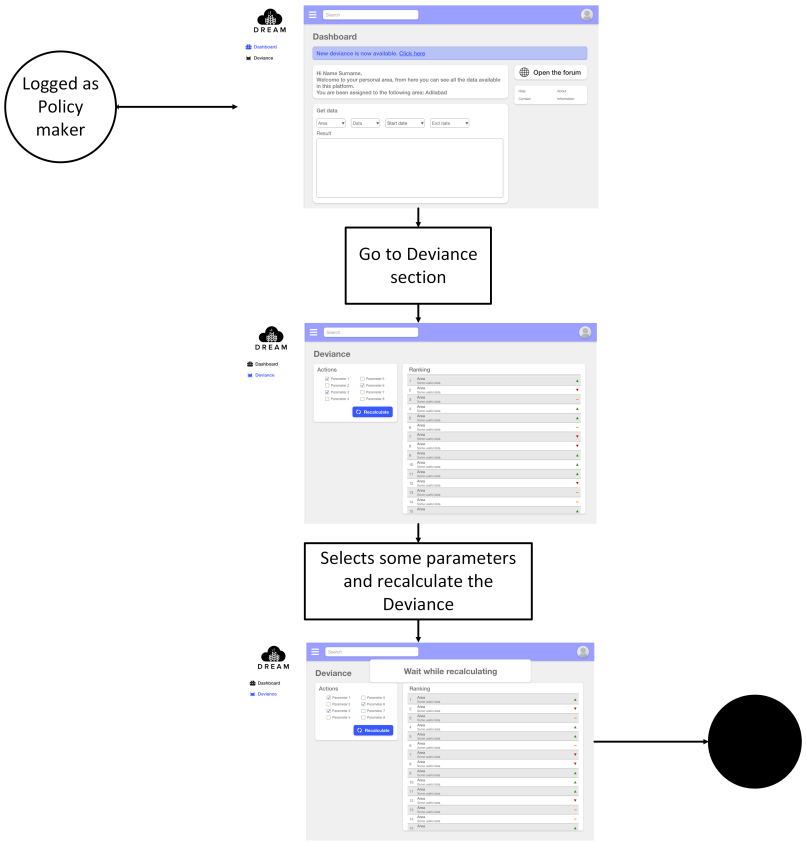


**Figure 30:** Policy maker recalculate the Deviance

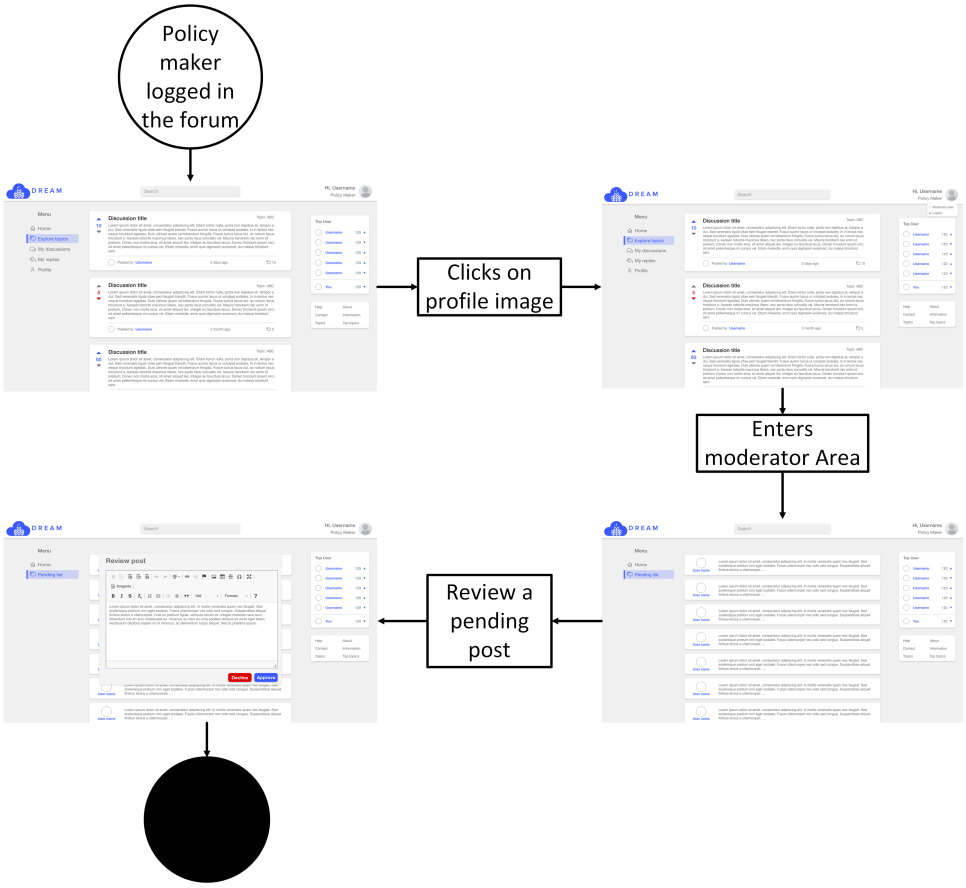


**Figure 31:** Policy maker approve a post in the Pending List

## 3.4 Forum interfaces

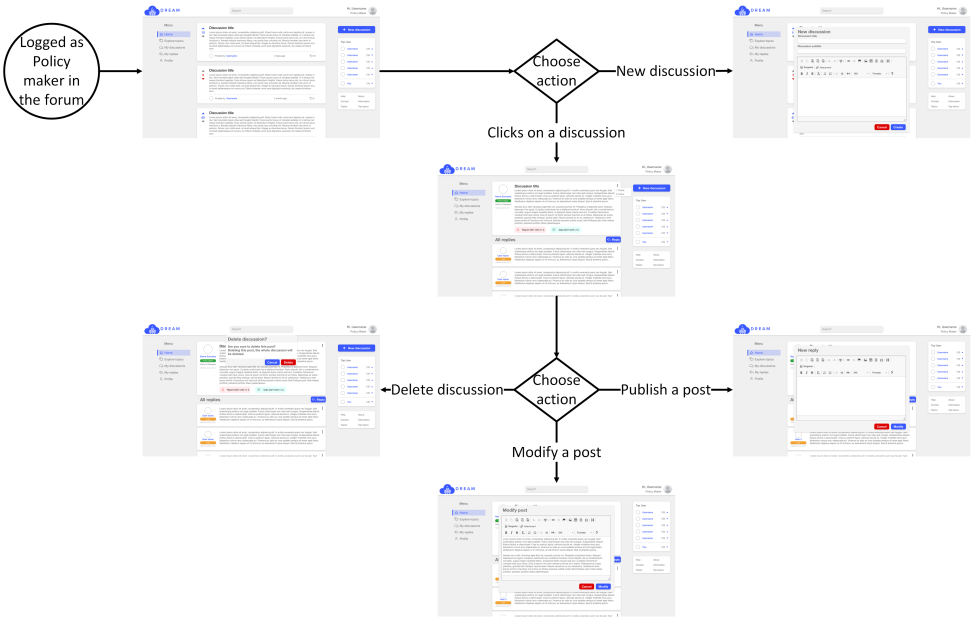


**Figure 32:** Policy maker Action in the Forum

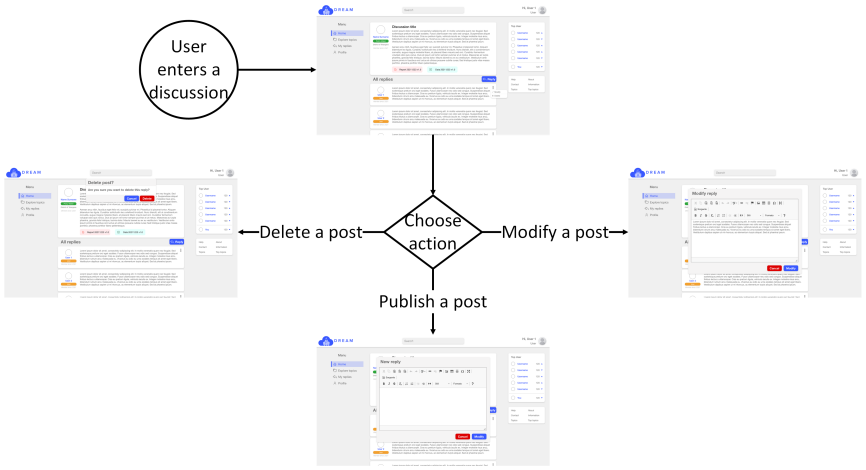


**Figure 33:** User Action in the Forum

## 3.5 Home interfaces

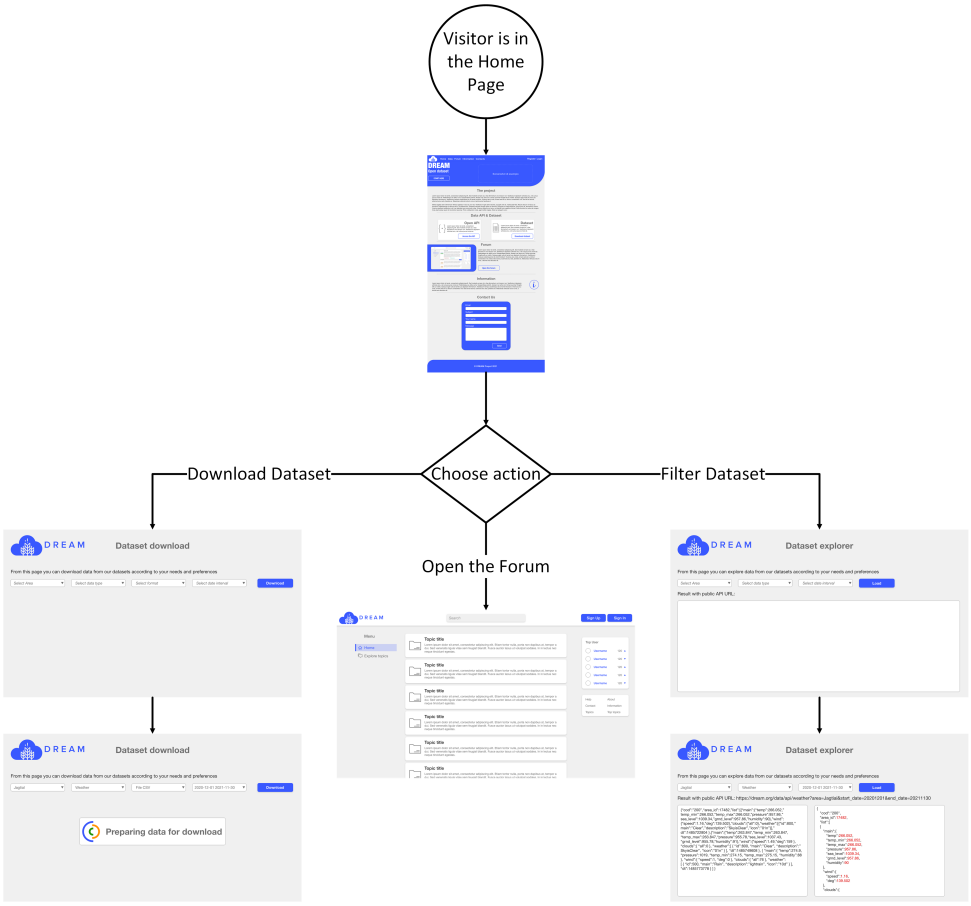


**Figure 34:** Visitor Action from the Home Page

# 4 Requirements Traceability

- **R1:** The system lets a Visitor to register to the identity provider.
  - **Authentication Manager**: in order to check that the User doesn't have an account.
  - **User Manager**: in order to insert the user into the system.
  - **Policy Maker Manager**: in order to validate the identity of a Policy maker.
- **R2:** The system lets identity provider's User to login in the correctly mapped role associated to its account.
  - **Authentication Manager**: in order to check if the User already has an account.
  - **User Manager**: in order to insert the user into the system.
- **R3:** The system lets a registered user to add a reply of a discussion.
  - **Discussion Manager**: in order to publish a post.
  - **Notification Manager**: if the post has been published by a Policy maker, in order to notify the new post publication to the one who follow the discussion.
- **R4:** The system lets the Policy maker to modify a reply in a discussion.
  - **Discussion Manager**: in order to modify a post.
- **R5:** The system lets the Policy maker to delete a reply in a discussion.
  - **Discussion Manager**: in order to delete a post
- **R6:** The system lets the Policy maker to see all the pending reply publication.
  - **Discussion Manager**: in order to retrieve the pending posts.
- **R7:** The system lets the Policy maker to accept a reply publication.
  - **Discussion Manager**: in order to accept a reply publication.
- **R8:** The system lets the Policy maker to decline a reply publication.
  - **Discussion Manager**: in order to decline a reply publication.
- **R9:** The system lets the Policy maker to create a new discussion.

  - **Discussion Manager**: in order to create a new discussion.

- **R10:** The system lets the Policy maker to delete a discussion.
  - **Discussion Manager**: in order to delete a discussion.

- **R11:** The system allows a User to modify the post written by him.
  - **User Manager**: in order to retrieve his replies list.
  - **Discussion Manager**: in order to modify a post written by himself.

- **R12:** The system allows an User to delete the post written by him.
  - **User Manager**: in order to retrieve his replies list.
  - **Discussion Manager**: in order to delete a post written by himself.

- **R13:** The system should send a notification to all the participant of a discussion when a change is made.
  - **Notification Manager**: in order to notify about a new post publication to the one who follow the discussion.
  - **Discussion Manager**: in order to retrieve the attendees of the discussion.

- **R14:** The system should send a notification to the User when his reply is approved.
  - **Notification Manager**: in order to notify a User when one of his post got published.
  - **User Manager**: in order to retrieve the information about the User.

- **R15:** The system should send a notification to the User when his reply is denied.
  - **Notification Manager**: in order to notify a User when one of his post got refused.
  - **User Manager**: in order to retrieve the information about the User.

- **R16:** The system lets the Administrator to add a new data source.
  - **Data Manager**: in order to add a new data source.

- **R17:** The system lets the Administrator to remove a data source.
  - **Data Manager**: in order to remove a data source.

- **R18:** The system lets the Administrator to modify a data source.
  - **Data Manager**: in order to modify a data source.
- **R19:** The system lets the Administrator to see the active data sources.
  - **Data Manager**: in order to retrieve the data sources.
- **R20:** The system lets the Policy maker to select a specific parameter when trying to recalculate the Deviance.
  - **Deviance Manager**: in order to select the parameters needed to recalculate the new Deviance.
- **R21:** The system lets the Policy maker to recalculate the Deviance.
  - **Deviance Manager**: in order to recalculate the Deviance.
- **R22:** The system lets the Visitor to search for a subset of the data.
  - **Data Manager**: in order to retrieve the data required.
- **R23:** The system lets the Visitor to download a subset of the data.
  - **Data Manager**: in order to download the data required.
- **R24:** The system lets the Visitor to navigate the forum.
  - **Discussion Manager**: in order to navigate the forum.

# 5 Implementation, Integration and Test Plan

The whole system is divided in two parts: the forum and the data aggregator. Both presents the same components that are:

- Client: the website which is load by the web browser
- Web Server (that also include the Service Provider Module)
- Application Server
- Internal Database
- External services such as IdP and the public datasources

These elements will be implemented following a bottom up approach, in order to have an easy implementation and testing. The main focus will be on the Application Server module because is the most complex part considering it is the core of the application and it also requires more testing.
The following table presents the main functionalities of the system, highlighting for each of them the importance for the costumer and the difficulty of its implementation.

**Table 1** Implementention and testing precedences

| Functionality | Module | Importance for customer | Difficulty |
|---|---|---|---|
| Sign up and Login | 6 | High | Low |
| Discussion | 1 | High | High |
| Notification | 1 | Low | Low |
| Downloader | 1 | High | High |
| Deviance | 1 | High | High |
| Data aggregator | 1 | High | High |

All the modules described in this document relies on a DBMS for each subsystem (one for the forum and another one for the data aggregator) that has to be implemented firstly.
According to table 1 we decide to implement the functionalities following their importance.

- **Sign up and Login:** the two functionalities are available in both subsystems and they are strictly related because of the adoption on the Identity Provider; in fact, a SignUp is the result of the first login. For that reason is a good choice to first implement the login function that also implies to set up the Shibboleth Service Provider Module and connect it to a test IdP and then implement the sign up method. The adoption of a the Service Provider module ensure that User attributes are available in the server session so a custom endpoint is

needed to read and save them in the DBMS for further uses.
In this parts is is also important to test the ability to save and retrieve User attributes from the DBMS correctly.

- **Discussion:** This functionality represents the core of the forum. It is implemented by means of the DiscussionManager which provides all the functionalities to create, modify and delete posts and discussions but also UserManager and PolicyMakerManager. In this part the DBMS cover a fundamental role because all the managed data have to be saved.
  The testing is done using automated test class. In fact a valid User and a valid Policy maker accounts are required, a discussion has to be created and some post added, modified and deleted by multiple and different controllers calls.

- **Data aggregator:** This functionality represents the core of the homonym sub-system. It is implemented by means of the DataManager and the AdministratorManager which provide the method needed to add, delete and modify data sources (by the Administrator) and methods to save, get and filter data from the DBMS. The testing is done using an automated test class for the same reason of the Discussion described above.

- **Deviance:** This functionality is strictly related to a Policy maker activity in the Data aggregator sub-system. It consist in the ability of using one or multiple data sets to calculate a ranking of a specific area using predefined parameters or custom ones. This is implemented by means of the DataManager (get and store data in the DBMS), DevianceManager (specific algorithm and parameter management) and the PolicyMakerManager which is used to get the Area in which the Policy maker works. The testing could be done by an automated test class but also a hand check is required to verify if the resulting ranking is coherent with the expected estimation in some more general cases.

- **Downloader:** It represents the principal service of the data aggregator subsystem which is responsible to connect to external data sources, fetch them and save the received content periodically. This function requires also some entities described in the Data aggregator to identity and parse the content correctly. Also the functionalities of saving data in the DBMS are provided by the Data aggregator.

- **Notification:** This functionality is implemented through the NotificationManager. To test the system will be necessary to use the Discussion functionality and verify that the system works either if a Policy maker approve the publication or if he decides to decline it. Different User will be needed to see if the notification functionality works also for the one who follows the discussion.

## 5.1 Clarification on component integration

In this section is present an overall description about how the components are integrated and how they communicate between each other.

### 5.1.1 Components Integration Forum

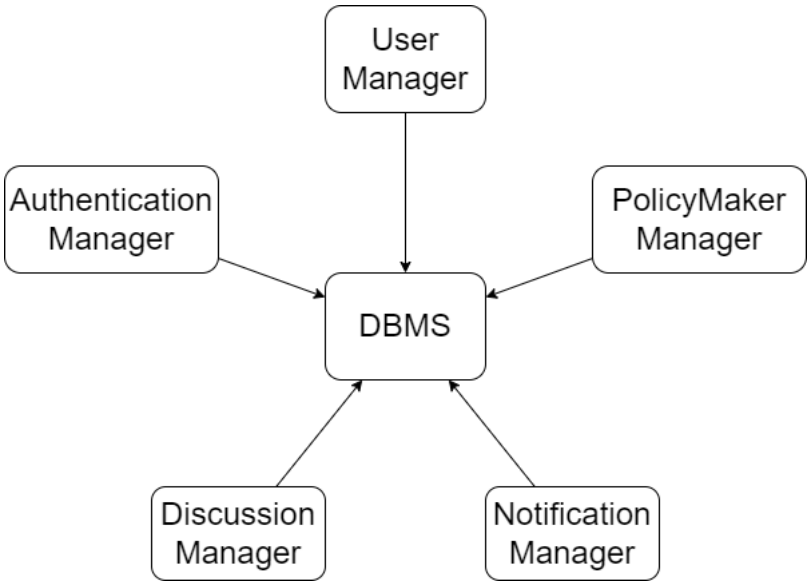


**Figure 35:** Components Integration Forum

### 5.1.2 Components Integration Data

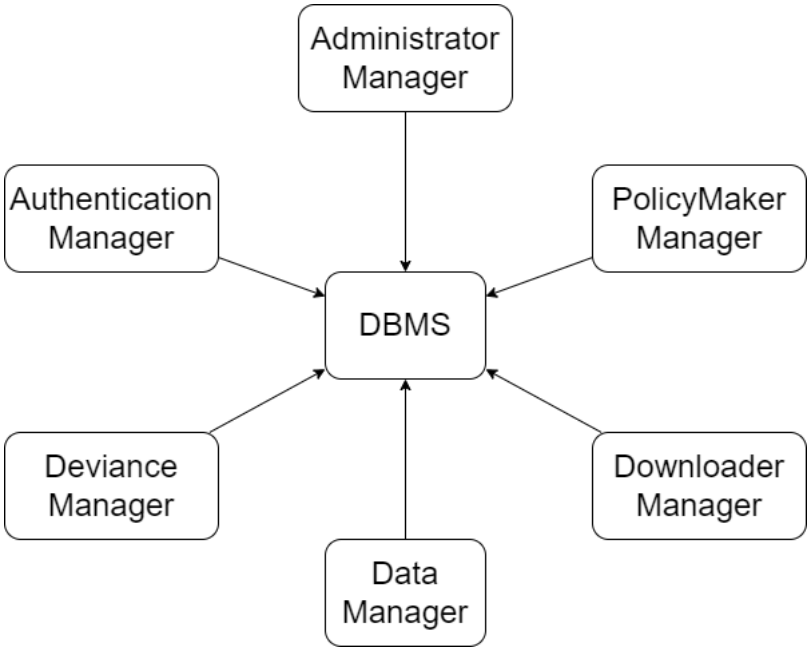


**Figure 36:** Components Integration Data

Using the bottom up method of implementation the first component to create is the DBMS. To go on, the main application component will have to be integrated with it.

### 5.1.3 IdP Integration


**Figure 37:** IdP Integration

The ability to use external Identity Provider requires an integration between the ServiceProviderModule with the AuthenticationManager by means of the Web-ServerModule. In fact a ServiceProvider (like Shibboleth) acts as a plugin for the web server to protect a resource. In our case the "..dream/login" path represents the resource to be protected that is the endpoint of the AuthenticationManager to complete the login flow.

### 5.1.4 ClientManager Integration Forum

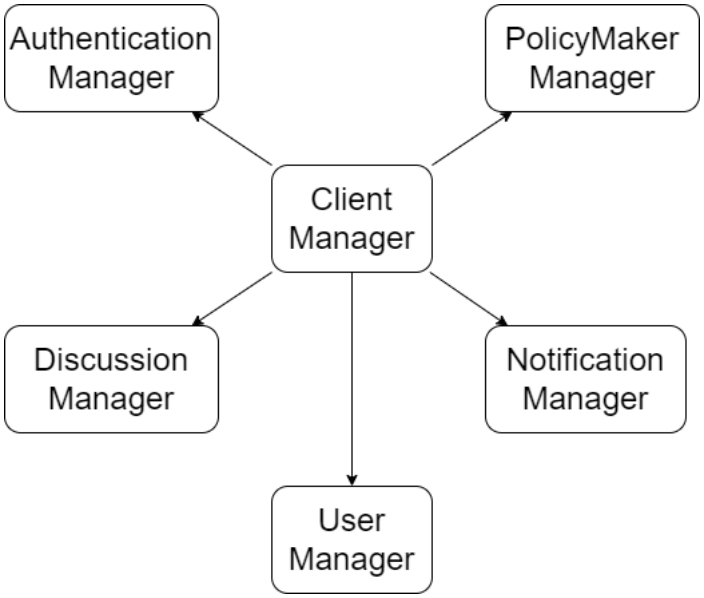


**Figure 38:** ClientManager Integration Forum

### 5.1.5 ClientManager Integration Data

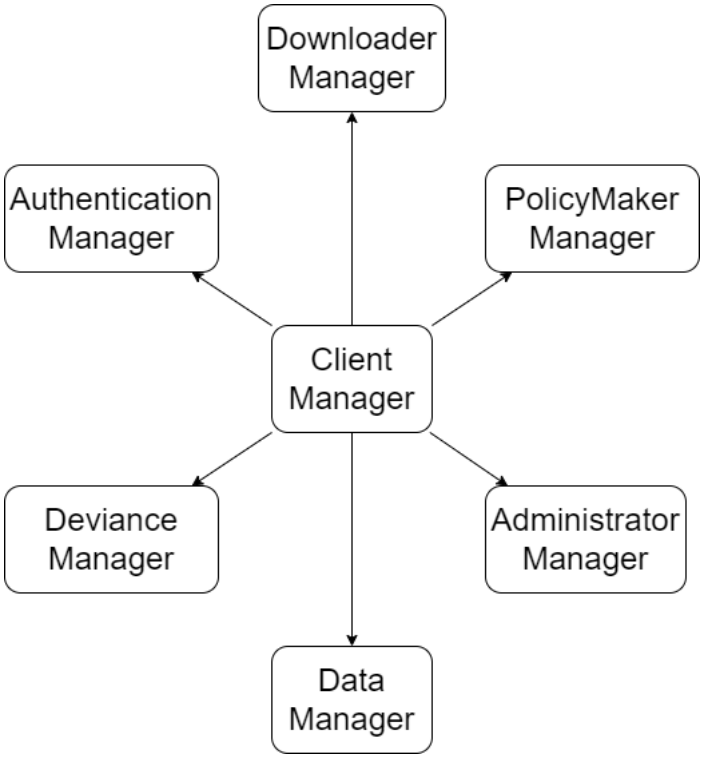


**Figure 39:** ClientManager Integration Data

Then, will be integrated the ClientManager for both the Forum and the Data sub-system, as shown correspondingly in figure 38 and in figure 39. Integrating those, will let the user to exploit all the functionalities they are allowed to.

### 5.1.6 Client-Server Integration


**Figure 40:** Client-Server Integration

Finally, it is possible to integrate the web server module and the web browser module, as shown in figure 40. This part brings the integration of the client-server communication.

# 6 Effort Spent

| | Alessandro Cecchetto | Mattia Siriani | Matteo Visotto |
|---|---|---|---|
| Time for S.1 | 2h | 1h | 1h |
| Time for S.2 | 10h | 12h | 8h |
| Time for S.3 | 2h | 4h | 6h |
| Time for S.4 | 2h | 1h | 1h |
| Time for S.5 | 2h | 1h | 2h |
| Total | 18h | 19h | 18h |