Semantic Web Project

1. **Problem statement**

Develop a small (web) application for pre-evaluating project proposals. The application is meant for financing bodies (e.g. UE), who have to finance some of the received proposals based on an evaluation process. Usually, each proposal is evaluated by 2/3 experts (randomly chosen from an IT system). In the end, all proposals which passed over a threshold are re-evaluated in a panel by other experts. The number of received proposals is extremely big and, of course, only a few of the received proposals are financed, so that a pre-evaluation would be helpful to fasten the process. Each financing call takes place within a program (e.g. Horizon 2020), which has different objectives: e.g. a program can choose to finance only the project proposing an IT system development.

1. **Scope and architecture**

The architecture used for the web application is REST which stand for **Representational State Transfer**.

**REST** is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations.

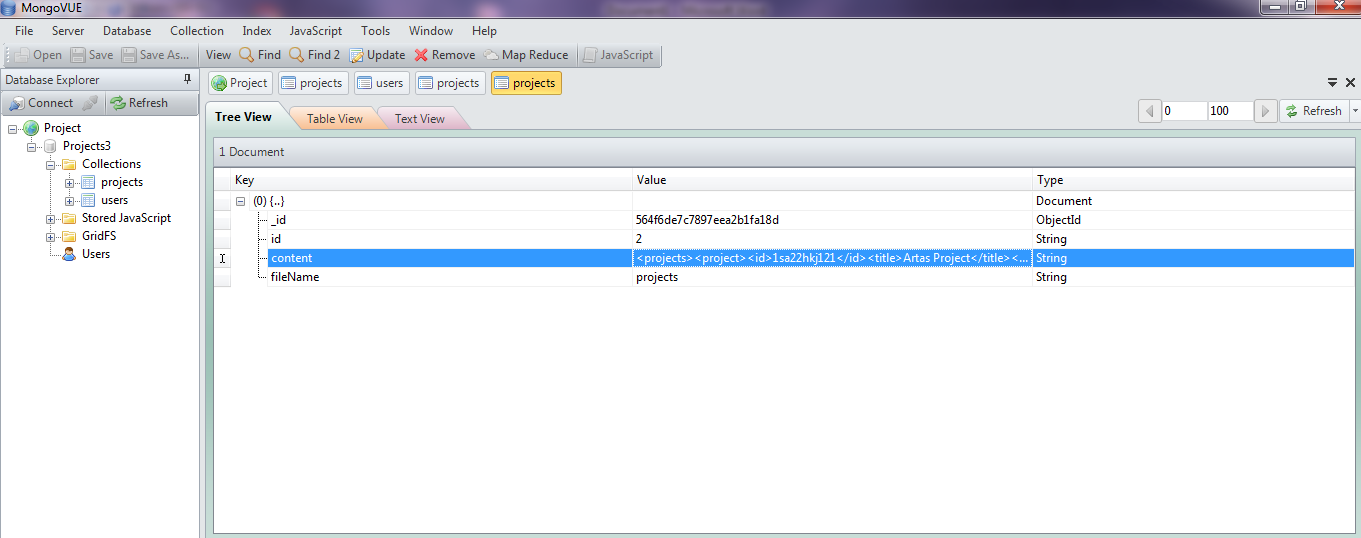
The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

The frontend content of the application was implemented using **AngularJS** which is a JavaScript framework designed for creating one-page web applications. Since it is a framework, it uses code templates and specific commands to perform special features such as filters and data-binding within HTML pages.

Angular offers flexibility to nicely separate presentation logic from business logic and presentation state.

The application was developed in **Spring MVC Framework** which is a popular open source application framework that can make Java EE development easier. It consists of a container, a framework for managing components, and a set of snap-in services for web user interfaces, transactions, and persistence. A part of the Spring Framework is Spring Web MVC, an extensible MVC framework for creating web applications.

Information regarding users and projects are stored using XML files in MongoDB ,a NoSQL Database.



The application is structured in two main parts:

1. Backend

For the backend, we used a layered arhitecture with access to the mongoDB database. Each of its layers had a specific functionality.

The Domain Layer is storing the information about how each object within our application should look and behave.

The Entity Layer is tied with the mongoDB database and provides persistence between the entity objects and their respective collections. (Entries in mongoDB are called collections).

The Repository Layer is the one that handles the user requests (coming from the frontend) and uses the Repository to get the requested data, transform it using the XML Converters (from the Util package) and send it back to the web application.

There is also a Utility package (not a layer) that is used for handling the XML data. It uses **DOMParser** for parsing the documents and create instances of their respective models, **XPath** to filter the data and perform queries on the xml. Also, there is a FileCreator used to create PDF, XML and XSLT files that are requested by the user.

1. Frontend

The frontend part is implemented using HTML5, CSS, Javascript with some specific frameworks (Bootstrap, AngularJs, Jquery, Ajax).

There is a main application (module) in which all the other controllers are hosted. Each controller is responsible of getting the data for its view. The view are resolved by using a route in the application. The view and the controller are tied together, due to Angular’s Two Way Data Binding.

1. **Use of XML**

For querying data from the XML documents we used XPath query language. The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria.

Using **XSLT** (Extensible Stylesheet Language Transformations) language, we transformed documents into HTML web pages displayed in the application.

Using **DOMParser** (Document Object Model Parser) we parsed the XML Files and transformed them into Java Objects.

Using **SAX** (Simple API for XML) we inserted new elements into the XML Files.