Proceedings of
the 29th International Business Information Management Association Conference

3-4 May 2017
Vienna Austria

ISBN: 978-0-9860419-7-6

Education Excellence and Innovation Management through Vision 2020:

From Regional Development Sustainability to Global Economic Growth

Editor

Khalid S. Soliman

International Business Information Management Association (IBIMA)

Automating Business Processes Using a REST Based Model Integrated with JQuery Technology

Dospinescu Octavian, Alexandru Ioan Cuza University, Iasi, Romania,
doctav@uaic.ro

Străinu Roxana-Marina, Alexandru Ioan Cuza University, Iasi, Romania,
poparoxi@yahoo.com

Strîmbei Cătălin, Alexandru Ioan Cuza University, Iasi, Romania,
linus@uaic.ro

Nistor Alexandra, Alexandru Ioan Cuza University, Iasi, Romania,
alexandra.anichitoaei@yahoo.com

Abstract

In this article we integrate different technologies (REST, JQuery, SOA) in order to automate some business processes that were already described in previous scientific papers. Starting from BPMN approach we build a rest based model for backend operations and finally we integrate REST API in JQuery Ajax calls in frontend operations.

So, starting from processes, the article proposes a complete integrated model (from backend to frontend) based on various modern technologies, passing from data to tables, operations, resources and json inputs and outputs.

Keywords : REST, JQuery, Technological Integration, Service Oriented Architecture

1. Introduction

In a previous scientific article there were studied the processes taking place inside universities and the existing information systems, or ideas of implementation (Strimbei, Dospinescu, Strainu, & Nistor, The BPMN Approach of the University Information Systems, 2016). Starting from business process modelling, and analyzing the business processes and the specific of the business or institution, we understood that the analysis phase must identify the requirements and organize the core processes taking place inside an university, which may become modules in a future system implementation or standalone applications, able to use available APIs to communicate with other related applications inside the future information system. Also, the case of Faculty of Economics and Business Administration from Alexandru Ioan Cuza University from Iasi, Romania, that has a number of tools involved in this system, which are not interconnected and work independently, rose the question how can we integrate existing data and new data so that the system flow is not affected by any technological change. The integration problem was also studied in a second article (Strimbei, Dospinescu, Strainu, & Nistor, RoaML FOR DATA INTEGRATION, 2016) which identified ROAML (Resource Oriented Architecture Modeling Language) as a tool for data integration and proposed a meta-model for this problem. Going further, other authors (Wixom, Dennis, & Roth, 2012) identified four important steps in the development of an information system: planning phase,

analyzing phase, design phase and implementation phase, each of them split into smaller steps. If in the first cited paper we have analyze and model one process, in this paper we will try to propose a model of implementation for modules or components (that may represent or serve one process) which are designed to facilitate data integration and communication over web, using REST API. Our REST API will be consumed by a JQuery script which may dynamically generate content on a page, based on the received data.

2. Literature Review

2.1. An Overview Over REST

Some authors suggest that the Representational State Transfer (REST) is not an architecture: it's a set of design criteria. You can say that one architecture meets those criteria better than another, but there is no one "REST architecture" (Richardson & Ruby, 2007). Based on this, we can somehow say that it actually represents a set of best practices in designing resources over which we can apply different HTTP verbs to obtain a specific response. Its creator however, states that REST architectural style for distributed hypermedia systems is a hybrid style derived from several of the network-based architectural styles (Thomas Fielding, 2000).

REST is also a lightweight way to build APIS (Application Programming Interfaces) to integrate or connect different software modules and different hardware devices, which has three major components identified in the literature review and which are very well represented in figure 1 below:

- Resources (URI – Uniform Resource Identifier)
- Verbs (GET, PUT, POST, DELETE)
- Representation (JSon, XML)

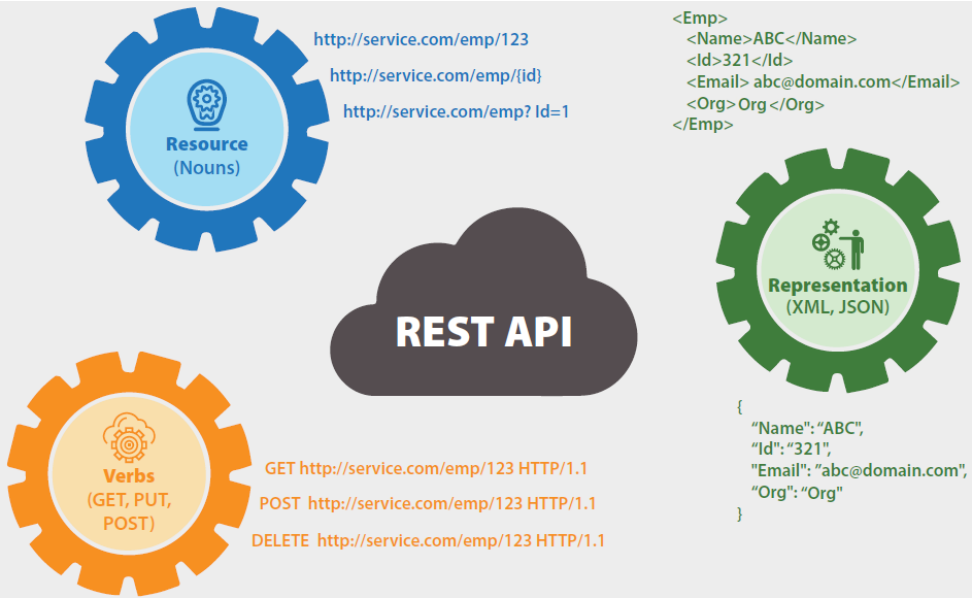

Figure 1: Simple REST API structure according to (Deepak, 2015, p. 3)

The main characteristics of REST are also presented by the creator of this paradigm in his public dissertation paper (Thomas Fielding, 2000):

- It's based on the client server architectural style, which improves the portability and scalability.

- It's stateless, meaning that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- It should implement cache constraints which require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- Uniform interface is the main difference between REST architectural style and other network-based styles. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolution.
- It's built based on a layered system style. Layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting. By restricting knowledge of the system to a single layer, we place a bound on the overall system complexity and promote substrate independence. Layers can be used to encapsulate legacy services and to protect new services from legacy clients, simplifying components by moving infrequently used functionality to a shared intermediary. Intermediaries can also be used to improve system scalability by enabling load balancing of services across multiple networks and processors. This layered style can be seen in figure 2 below.

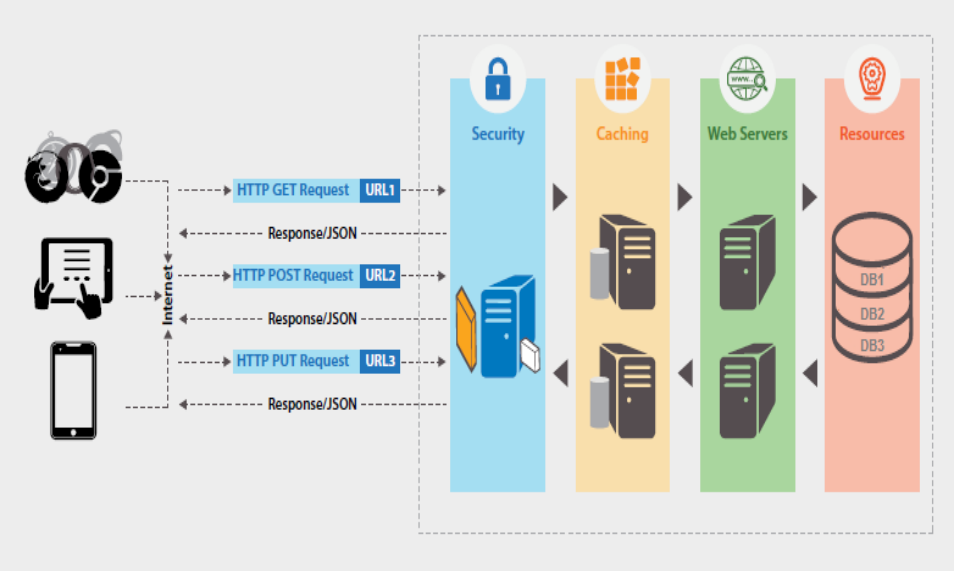

Figure 2: REST layered style according to the model proposed by (Deepak, 2015, p. 11)

When we talk about REST, we refer to specific APIS designed to expose services. By services, in REST, we understand the response we receive and we need to use it in one or more application which need this specific data, or the data we send to the server using specific actions. The response is received or sent via an URI and using a HTTP method.

APIs used to mean low-level programming code interfaces. In recent years, the term has been re-appropriated to mean simple interfaces provided over HTTP. Typically, it equates to REST interfaces, which provide data by using the JSON data format (sometimes XML), and the HTTP

verbs PUT, GET, POST, and DELETE to depict create, read, update, and delete actions. These protocols and data formats are simpler to use than the web services standards based on SOAP that were more prevalent in early SOA. Also, they are more suited to such languages as JavaScript that are commonly used when making API requests (Clark, 2016).

2.2. SOA and REST – short review

The architectural terms of Service Oriented Architecture (SOA) and Resource Oriented Architecture (ROA) emerge in this context, and some differences occur. From the implementation perspective, SOA is mainly based on SOAP (Simple Object Access Protocol) while ROA is built using REST. From a modelling perspective, authors argue that the SOA is supported by SOAML modelling paradigm, while ROA has no modelling equivalent, and they propose some guidelines for ROAML (Olaru & Strimbei, 2015). The same authors found some differences between the ways of implementation from both architectural approaches like: service coupling, service composability, service discoverability and service contracts, which may be ones of the major advantages of a REST architecture (Olaru & Strimbei, 2015).

Other author (Deepak, 2015) identified some specific advantages of REST over SOAP (except coupling which is common):

- SOAP uses only XML for messages. REST supports different formats.
- REST messages are smaller in size and consume lesser bandwidth.
- REST is better in terms of performance with better caching support.
- No third party tool is required to access REST web services. Also with REST-based services, learning is easier when compared to SOAP.
- There is less coupling between REST Clients (browsers) and Servers; feature-extensions and changes can be made easily. The SOAP client however, is tightly coupled with the server and the integration would break if a change is made at either end.

To avoid polemics over architectural styles, we can state that REST may be very well a tool to implement services. This means that SOA can be implemented using both SOAP and REST APIs. The differences occur at more specific levels of implementation and best practices.

The difference between SOA an APIS (because REST became a tool to build APIs) is a starting point to understand what each of them represents, and if they are related:

- In SOA programs, service exposure was about exposing each business function so that it could be reused as much as possible. This way, each new project didn't have to go through the pain of performing integration to the back-end system again. The typical consumers were internal applications that attempted to put fresh user interfaces onto older systems of record.
- APIs had a different starting point, with the assumption that integration was already simplified. They are designed for the context in which they are likely to be used. For example, they are ideally suited to provide the data that is required by a particular type of mobile application (Clark, 2016).

With mobile devices evolution, APIs gained popularity because they offer simple access to back-end functions and data. In the same time, the focus has been on APIs as something to expose publically to external consumers. The lines between SOA web services and API are now blurred and almost irrelevant. They have differences in their origin, to whom they are exposed to, and the data models they use, but many SOA "services" could also be potentially described as internal APIs. (Clark, 2016)

We can extract from here a simple conclusion, which represents the major advantage of SOA implementation using REST APIs, and this is the ease of data integration. While many institutions or companies already have some software solutions for internal business processes, some legacy, some exposing web services, REST API becomes the link between any other new application and the existing ones. Another advantage is the use of this kind of API to implement backend operations, while the front-end, or the user interaction can be distinct and independent. Because any major programming/scripting language has developed libraries to help implementing this type of API, it supports a wide range of technologies. In the same time, many technologies are prepared to consume REST. Starting from these advantages and the ones identified by Deepak, we can start building our model.

3. Building a REST Based Model for Backend Operations

While some authors argue over architectural styles and try to integrate REST in architectures like: SOA, ROA or micro-services, one thing is clear: REST APIs expose services which may be consumed by clients. The mechanism is simple and based on 3 main and mandatory components which the resource is programmed to offer:

- An URI
- A response
- A HTTP method

The purpose of this paper is to build a REST based model which uses/consumes REST APIS (services) to send/receive data to/from the server using JQuery. The technology used to build REST APIS is Java and JAXRS. It may be very well implemented using Python, Ruby, PHP.

We want to explain how the REST API is easily integrated in web apps and not only. The consumer (or the client) will be represented by JQuery scripts which implement AJAX calls to access the resources (which will return the response from the server). It may be very well represented by a mobile app, or a Java app.

The resources will have to access a database, and to perform CRUD operations over its tables. The consumer (in our case JQuery script) will only use the URI of the resource and the desired method to get or send data to and from the server.

We have to mention that when a resource is build:

- It doesn't hold the effective data
- It already has attached a specific HTTP method, predefined in implementation
- It produces the desired type of data (JSon or XML)
- When accessed, returns or sends the data in the desired format (meaning that it performs the effective CRUD operation over the database)

This is very useful to help the reader understand that the REST API doesn't trigger itself. It always needs a trigger represented by the consumer, to perform the desired operations. The trigger may be any application or script which can perform HTTP action and can parse JSon or XML data formats.

First, the main step to realize what we proposed is to create a class for each existing table from the database. This class will contain the fields of each table defined as attributes, and the getters and setters for each attribute. Another class, will use instances of these classes to perform operations over the database (the effective CRUD operations), while a third class will use REST annotation to create the effective resources. We may consider that the operations into the database performed by the second class are the effective services or operations, while in the third class we define the REST model and we decide which services we will call.

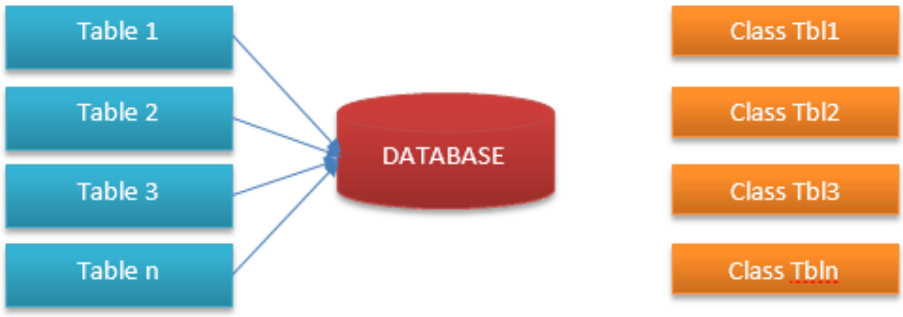

Figure 3: The tables from the database and the corresponding classes

For the moment, when we created only the corresponding classes for each table, there is no connection and no possibility to perform any operations over the tables from the database, as we can see in Figure 3 above. The next step is to implement a class which intermediates the connection with the database and the CRUD operations. We can call it service class, operations or actions class. To be aligned with the paper theme and to avoid misinterpretation, we will call it operations class. Each operations class (or service class) will contain an element of type Tbl1, Tb2, Tbl3 or Tbln, depending on the table is serves and will perform operations over the corresponding tables as we can see in figure 4 below.

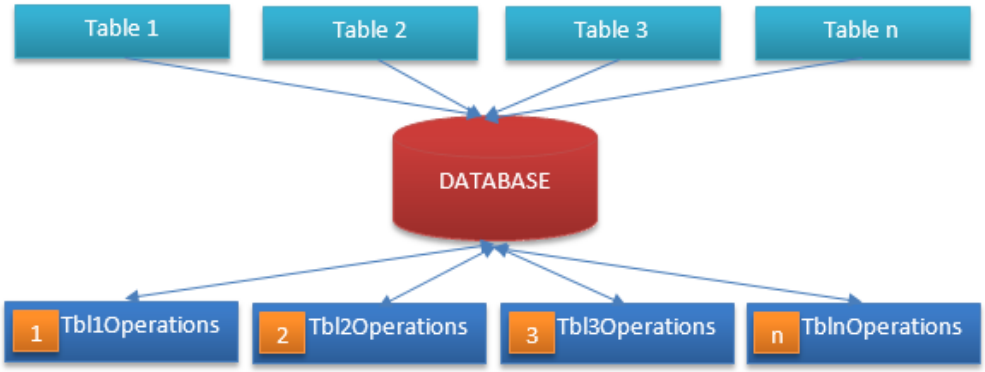

Figure 4: The tables from the database and the classes that perform operations into the database

Finally, the REST implementation will create the API we need for data integration, as it was also stated in a previous paper (Strimbei, Dospinescu, Strainu, & Nistor, RoaML FOR DATA INTEGRATION, 2016). We can use this data in a web page using jsp, or JQuery, or in Android mobile app built using Java, but not only in this cases. There are many other types of web applications which use and consume services offered by REST APIS. In our case, our resources will produce and consume JSon, which is easy to use by JQuery, which will be responsible to dynamically generate content into the page using data from the database and the user actions. Similar, an Android mobile app can use this data to feed a Fragment into an interface.

In JAXRS, the 3 main components of REST API are implemented using annotations as we can see below:

- The resource: using @Path and @PathParam
- The representation: using @Produces and @Consumes
- The verb (HTTP action): using @GET, @PUT, @POST, @DELETE

Using a @QueryParam annotation, the developer can suggest to an implemented method that it will have a variable in the resource URL that will use it latter in specific implementations inside that method.

Finally, our model will have a conceptual model represented in figure 5 below, and while a REST API for a table called Users is described in the next lines of code.

```
Public Class UserResource(){
@Path("/users")
@Consumes({MediaType.APPLICATION_JSON})
@Produces(MediaType.APPLICATION_JSON)
public class UserResource {
private Map<String,User> users = new HashMap<>();
private UserService usersService = new UserService();

   public UserResource() throws Exception{
            try{
                     users = usersService.readUsers();
            }catch (Throwable ex){
                     System.err.println("Uncaught exception - " + ex.getMessage());
       ex.printStackTrace(System.err);
            }
   }

   @GET
   public ArrayList<User> getAllUsers() throws Exception{
            try{
                     users = usersService.readUsers();
            }catch (Throwable ex){
                     System.err.println("Uncaught exception - " + ex.getMessage());
       ex.printStackTrace(System.err);
            }
                     return new ArrayList<User>(users.values());
   }

      @GET
      @Path("/{user}")
   public User getUserInfo(@PathParam("user") String userID){
            //User u=new User();
            return usersService.getUserData(userID);
   }

   @PUT
   @Path("/{user}")
   public void updateUser(User user){
            usersService.updateUserData(user);
   }

   @DELETE
   @Path("/{user}")
   public void deleteUser(@PathParam("user") String user){
            usersService.deleteUserData(user);
   }

   @POST
```

```
    public void insertUserData(User user){
            usersService.insertUserData(user);
    }
}
```

We can see from the code that the resource contains a service Object called UsersService, and in the case of our scheme (from figures 4 and 5) may be called UserOperations. The service object (or the operations object) is the one that performs the operations into the database. The REST API (built in UserResource class) only calls the methods from that class and contains the REST annotation we need to have for a valid API over http.

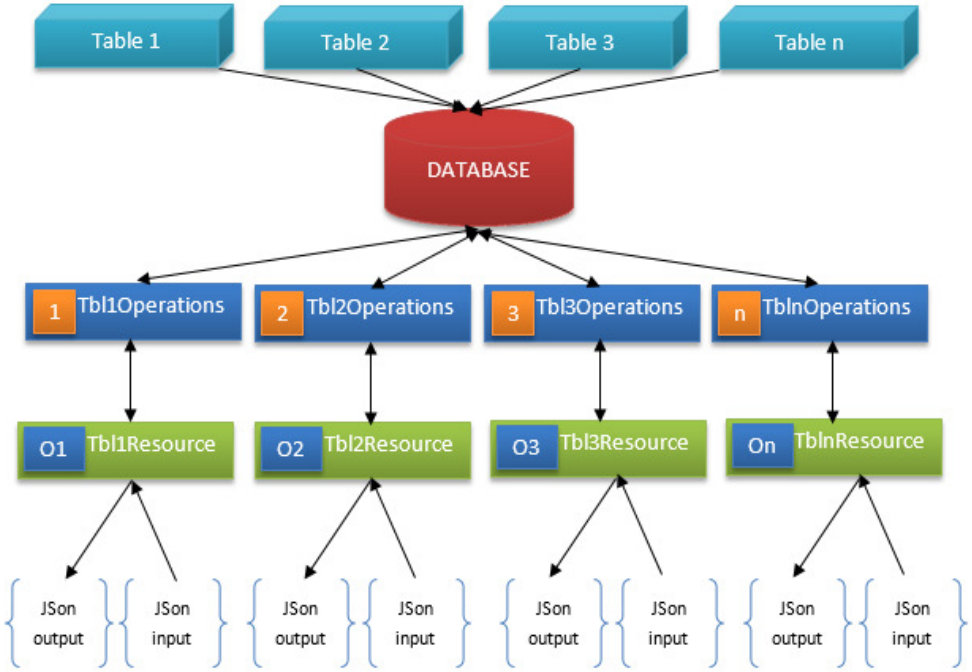

Figure 5: The resources which use the Operations classes to communicate with the database and which produce and consume JSON

When we look at the model above, we can understand some important things. The business logic is completely separated by the consumer of the data. Also, in our case, our resources don't perform the effective operations into the database, they call specific methods from the Operations classes (which we can say that offer services in our REST model). Starting from this idea of separation of the business logic from the effective data, we can say that we have built a simple MVC model in which, the model is represented by the classes of the tables, the controller is represented by the classes which perform operations, while the view is represented by the REST APIS which just return and consume JSon data.

It can be easily noted that this API represents just the backend of an application, while the frontend is treated by a different technology. This technological separation also offers another plus to data integration.

4. Integrating REST API in JQuery AJAX calls

We choose JQuery, because it's a JavaScript library which communicates very well with the frontend part of a web application and it doesn't need any framework to function. Being a library, it comes equipped with some standard operations and functions, one of them being the AJAX calls necessary to access the backend scripts which communicate with the server.

In the context of this paper, the consumer, in our case an AJAX call from a JQuery script, has a standard form with 3 main components:

- The request type (which may be only GET for receiving data from the server, or POST to modify data on the server, or to protect sensitive data sent in URL)
- The URL of the script which produces the desired output, or the script that consumes the data this script generated and sends it to the server
- The success function which uses the response from the resource and acts accordingly

Also using contentType attribute inside an AJAX call we can announce what type of data we are sending to the server, while using dataType attribute we know what kind of data we are expecting from the server.

If we think about audience of our API, is better to think before implementing the API, what is specific for each consumer. In our case, because an AJAX call in JQuery is limited only to GET and POST type of actions, we may also consider to differentiate the resources for each type of C_UD operation.

So the simulation of an AJAX call using a resource created above may be as follows:

```
$.ajax({
        type: "GET",
        url: myResourceURL,
        dataType: "application/json",
        success: function(response){
        //the actions we can make using the requested data which resides in the response variable
        }
});
```

In this call, myResourceURL can be any link defined in our rest APIs, and which has annotated the GET method, and a path like /myResourcesDomain/Users (to get data for all users) or like myResourcesDomain/Users/userID (to get the data for a specific userID), supposing that Users is one of the tables stored into the database.

The response for this type of call like myResourcesDomain/Users/userID may be a JSon with the following format: {"Username": "userID","FirstName": "MyFirstName","LastName": "MyLastName"}. This data may be dynamically loaded into a HTML page, by generating HTML content from the JQuery script.

The final scheme of this model is represented in figure 6 below:

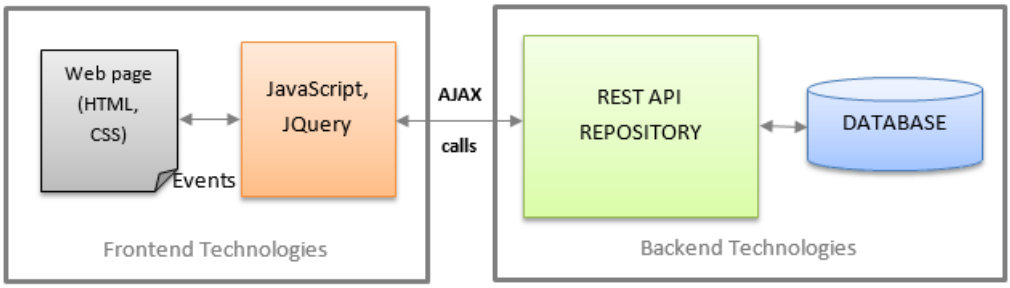

Figure 6: REST API integrated with frontend technology using AJAX and JQuery

The fact that we use resources (URIs) instead of script names to access the data we need from the database, offers not only an elegant solution to write code, but a much easier way to track the backend sources, to organize code, and even justifies the fact that modelling for ROA may be an important starting point in designing a and modelling components in a resource based SOA. The repository of REST API represents the core of the backend and it may also contain the business logic (the operations from the database), not only the resources from the figure 5 above.

The separation of backend and frontend, allows the use of different technologies to consume the resources. The name API, just states the data integration and correlation, and this is why we can say that REST is the perfect API for web-based applications.

In the left side of the scheme, we can also add mobile technologies to be integrated with our REST API repository, using specific HTTP calls or requests.

5. Conclusions

In this article, our contribution consists in developing a complete integrated model (from backend to frontend) based on modern technologies (jQuery, Rest, SOA), proving that the integration process can be a successful one. It is very important to understand that the business processes are the starting point of the integration and the technologies are only the support of the information integration. Although we used jQuery Ajax in the frontend section, it is obvious that the client-side could be implemented in any technology that allows calls to the server-side.

Acknowledgments:

„This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-II-RU-TE-2014-4-0748”.

References

Clark, K. (2016, January 21). *Microservices, SOA, and APIs: Friends or enemies?* Retrieved January 30, 2017, from http://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

Deepak, K. (2015). Best Practices for Building RESTful Web Services. Infosys Limited.

Lin, G. C., & Desmond, K. E. (2012). Service Oriented Architecture. In *A Fresh Graduate's Guide to Software Development Tools and Technologies* (pp. 2-31).

Olaru, G., & Strimbei, C. (2015). Modeling Web Services with RoaML. *Economy Informatics vol. 15, no. 1*, 26-34.

Richardson, L., & Ruby, S. (2007). *RESTful Web Services.* Sebastopol: O'Reilly Media.

Strimbei, C., Dospinescu, O., Strainu, R., & Nistor, A. (2016). RoaML FOR DATA INTEGRATION. *IBIMA* (pp. 2782-2791). Seville: Ibima.

Strimbei, C., Dospinescu, O., Strainu, R., & Nistor, A. (2016). The BPMN Approach of the University Information Systems. *Ecoforum*, 181-193.

Thomas Fielding, R. (2000). *Representational State Transfer (REST)*. Retrieved January 22, 2017, from http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Wixom, B., Dennis, A., & Roth, R. (2012). *System Analysis and Design, Fifth Editioin.* John Wiley & Sons, Inc.