

# Architectural pattern for service collaboration

Agon Memeti<sup>1</sup>, Betim Cico<sup>2</sup>

<sup>1</sup>Department of Computer Sciences, Faculty of Natural Sciences and Mathematics, University of Tetova, Tetovo, North Macedonia

<sup>2</sup>Department of Computer Engineering, Metropolitan University, Tirana, Albania

## Article Info

### Article history:

Received Jan 13, 2022

Revised Jun 10, 2022

Accepted Jul 17, 2022

### Keywords:

Flexibility

Interoperability

REST architecture

Service reuse

Web applications

## ABSTRACT

The aim of this paper is to propose a modeling framework, tailored to build efficient, elastic and autonomous applications from tasks and services. It includes integrated services to develop the software products, reusing on demand in-house services with specific requirements and flexible the representational state transfer (REST) services. The idea is to decouple authorization for reduced service dependency and to provide a possibility for developing the whole application by increasing the existing application flexibility. Based on the fact that there are different web application platforms that serve to offer services to users but they are not integrated; we propose a framework with high flexibility degree, especially integrating the most used services such: e-learning, administrative, and library services, as University services are concern.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



## Corresponding Author:

Agon Memeti

Department of Computer Sciences, Faculty of Natural Sciences and Mathematics, University of Tetova

Street Ilinden, Tetovo, North Macedonia

Email: [agon.memeti@unite.edu.mk](mailto:agon.memeti@unite.edu.mk)

## 1. INTRODUCTION

The new economy is marginalizing the value of legacy software applications. Traditional software involves significant up-front acquisition costs and requires recurring maintenance and support. Organizations soon find that while traditional software can be customized, it often leads to version-lock and the inability to preserve changes through an upgrade. Simple upgrades become costly, resource-intensive reimplementation projects [1]. The discipline of service integration is quite new and in the focus of many research groups in different institutions and around the world. It is presented during different software's and application implementations and the idea come for something new, innovative, and contemporary around the world. In addition, it offers a new possibility of implementing web applications reusing existing applications providing data accessibility of existing applications without having to request permission from the systems administrators within the company.

The traditional e-services (applications), (such university services) as our case study are different web application platforms that serve to offer services to users but they are not integrated. They are isolated in psychical and business process aspects, and require specific authorization for rebuilding. In addition, the existing system has several problems such as lack of mobility, accessibility, service flexibility and portability apart from the necessity of physical presence of the administrator to ménage authorization when rebuilding applications (services) is concerned.

As service integration is the biggest problematics and challenge which tends the use of software or computer system architectural principles by linking in-house services (applications) of a single organization in order to share data, exploring how we can incrementally integrate existing web applications to use the same data, by exporting all the applications as simple services. While exporting them as basic services, the next problem is service authorization. We are resolving it using our proposed framework.

The proposed and implemented framework provides integration of all in-house (existing) university e-services by proposing a model/framework for web-based application (services) integration. This model will reuse the existing services (applications) achieving reduced service (application) dependency by decoupling authorization framework and providing integration of registered in-house services such as university services. This will lead to, increasing service flexibility; preserving stability of services (applications) and increasing portability and service (application) interoperability. The paper is composed in four sections, starts with background, proposed framework/model, its implementation/results and a conclusion.

## 2. BACKGROUND

### 2.1. RESTful services

Representational state transfer ful (RESTful) services are an architectural style (collection of principles), lighter than simple object access protocol (SOAP)-based web services, due to their simplicity, heterogeneity and web-based format. Entities/resources are identified by unique universal resource locator (URLs). Including how resource states are addressed and transferred over hypertext transfer-transfer protocol (HTTP) by a wide range of clients written in different languages. A RESTful design constrains web architecture for the purpose of simplifying usage, development, and deployment to the web. Firstly, this design requires the use of a client-server architecture that separates the user interface from the data storage concerns, and the biggest benefit is that development of components can proceed independently. The stateless constraint requires application state to be maintained exclusively by the client [2].

The REST architectural style contrasts the remote procedure call (RPC/SOAP) architectural style, since the great advantage of the REST, compared to other web services technologies, is that the message exchanged is transmitted directly over the HTTP protocol without encapsulation need and without use of envelopes. In this architecture, the focus is on resources and not on the call to the procedure/service. This approach is interesting for applications where the focus on interoperability is greater than the formal contract between the parties [3]. Fielding [4] REST architectural style describes six constraints which define the basis of RESTful-style: i) the uniform interface is the interface between clients and servers, simplifying the architecture, enabling each part to evolve independently; ii) stateless, as the key of REST services, refers to the necessary state to handle the request whether as part of the uniform resource identifier (URI), query-string parameters, body, or headers; iii) cacheable, as clients cache responses, eliminating some client-server interactions further improving scalability and performance; iv) client-server-clients not concerned with data storage, which remains internal to each server, so that the portability of client code is improved and servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable; v) layered system-a client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way and code on demand; and vi) the optional constraint [5].

These operations are defined directly in the HTTP method and are also known as HTTP verbs. In the literature, there is a common association with the acronym create, retrieve, update, and delete (CRUD) which means creates (POST), read (GET), update (PUT), and delete (DELETE). Create to add new entries, read to retrieve existing entries, update to update existing entries and delete to remove entries [3].

### 2.2. RESTful service components

RESTful service includes the following components: i) RESTful service provider, it is the software system that processes the request and provides data; ii) RESTful service client, the client that accesses the RESTful service; iii) resource, fundamental concept of RESTful service; iv) URI, the URI that identifies a domain resource; v) uniform interface, simplifies and decouples the architecture as a fundamental part to the design of any REST service; and vi) HTTP, RESTful systems communicate over HTTP verbs (GET, POST, PUT, and DELETE).

## 3. PROPOSED ARCHITECTURAL PATTERN

Since all REST traffic is HTTP using the standard operations, it is easy to view the data returned by the services using a web browser, which is very useful when debugging. In addition, the short development time and large freedom when coding, allows for a short time span from idea to implemented system as an important factor [6]. The idea is to integrate the existing services by exporting applications as basic services using open authorization (OAuth 2.0) [5], [7]–[11] and REST services with the aim of resolving service authorization since while exporting applications as basic services the next problem that needs to be resolved is service authorization. Using our proposed framework clients will request access token using its client credentials when the client is requesting access to the protected resources under its control.

A client application will need a separate token for each service it accesses. Otherwise, a service might misuse the clients token to access another service for which only the client has been authorized.

Another service may also be considered as a client application (clients are also services) [2], [6], [12]–[14]. Each service should provide a possibility for developing the whole application before even being given any permission. Once the application has been tested, then we may require the permissions and switch to the real endpoints, with these steps: i) endpoints should be versioned for greater flexibility. It should be possible to deploy newer versions without affecting the existing services; ii) discuss the scaling (load balancing and similar) issues. Leaving this responsibility to the services it might be more flexible; iii) the proposed sequence of execution is narratively explained in the steps below and further graphically represented in the diagram in Figure 1; iv) the client(s) requires tokens to the coordinator for accessing one service (or more services); v) the coordinator generates tokens (and saves them) and returns them to the client. One token is valid for one client, for many services, as defined in the roles list of the coordinator; vi) the client makes a request to a service using the token he has received for that service from the coordinator; and vii) before granting the execution to the client, the service verifies the token by calling the coordinator.

Figure 1 represents the sequence diagram [15], where the client initiates the request to the centralized coordinator for service access. The client includes its client identifier, requested service and the local state. The coordinator authenticates him and sends him back, once access is granted (or denied), based on the request. In order to speed up the step 4 and avoid multiple calls to the coordinator, the service can also locally cache the token lifetimes, thus allowing multiple requests by client without invoking the coordinator at all. Firstly all services should be registered to the coordinator in order to get access token for authentication.

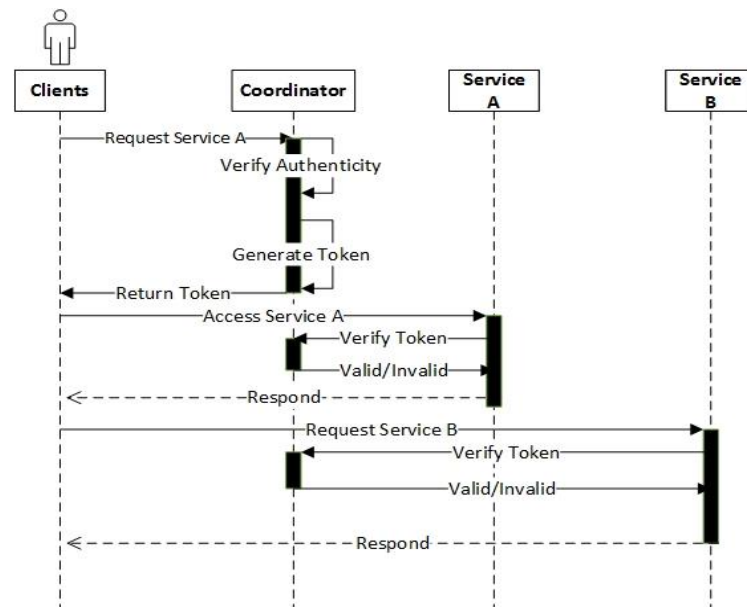


Figure 1. Proposed framework for flexible REST services-sequence diagram

### 3.1. Coordinating services

The proposed authorization flow between two communicating services and the coordinator handling the authorization between them is defined below. Note that in this architecture, the trusted party is the coordinator and each service delegates its authorization concerns to the coordinator. As a prerequisite, each service should be registered in the coordinator. We leave out the aspects of registration and generation of authentication credentials, since these aspects are not relevant to the coordination. The only important aspect is that each service disposes its own credentials which are used to authenticate the service when it accesses the coordinator. We illustrate this authentication using basic authentication principles, but any relevant authentication method can be used in this context. In addition, tokens and similar data are abbreviated for easier reading. Figure 2 depicts the authorization flow in a case when service 1 (A) wants to access a resource in service 2 (B):

- a. In order to access a resource, the client services (A) need to acquire a token from the coordinator. Since the access to each service requires a separate token, the service for which we require a token should be specified in the request.

```
POST /tokens HTTP/1.1
Host: coordinator.example.com
Authorization: Basic MzRoamdzZGY3NmczND...
...
{
  "service": "B"
}
```

- b. If the service 1 (A) is authorized for the requested resource, the coordinator will either create or return an existing token, together with the time of validity.

```
HTTP/1.1 201 Created
...
Expires: Sat Mar 21 15:14:09 +0000 2015
...
{
  "token": "802057ff762hdk3936...",
  "expires": "Sat Mar 21 15:14:09 +0000 2015"
}
```

- c. If the service 1 (A) is not authorized for specific service that was requested, the response will be as follows:

```
HTTP/1.1 403 Forbidden
...
{
  "reason": "Access to this service is not allowed "
}
```

- d. After having retrieved a valid token, service 1 (A) requires a resource from the service 2 (B) by authenticating using the previous token:

```
POST /courses HTTP/1.1
Authorization: Token 802057ff762hdk3936...
...
{
  "code": 1234,
  "name": "Service Oriented Applications",
  ...
}
```

- e. The service 2 (B) validates the token by calling the coordinator, together with information about the concrete required resource.

```
GET /authorization HTTP/1.1
...
{
  "token": "802057ff762hdk3936...",
  "method": "POST",
  "resource": "/courses"
}
```

- f. It is the coordinator's responsibility to decide whether service 1 (A) is allowed to access this particular resource on service 2 (B). If not authorized, it will send back a code 403 response:

```
HTTP/1.1 403 Forbidden
...
{
  "reason": "You don't have the necessary permission to access this resource."
}
```

- g. If not authorized, service 2 (B) should transfer the same response to the service 1 (A).  
h. If the service 1 (A) is authorized, the coordinator returns a 20× response.

```
HTTP/1.1 200 OK
...
```

- i. In this case the service 2 (B) should continue with processing the original request and eventually return the corresponding response.

```
HTTP/1.1 201 Created
...
{
  "id": 89,
  "code": "1234",
  "name": " Service Oriented Applications",
  ...
}
```

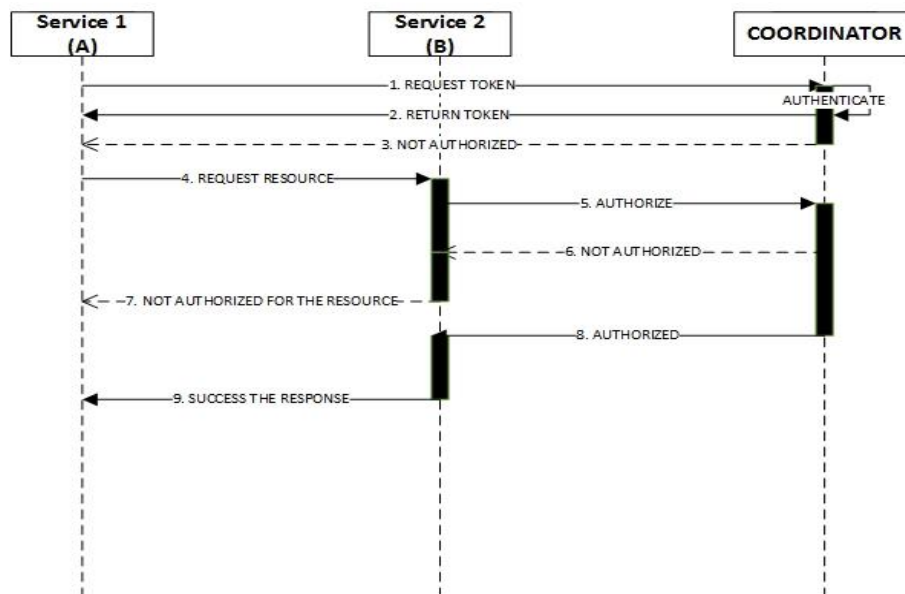


Figure 2. Coordinator REST interface-protocol

As we can see in the previous flow, the coordinator can define different policies for service access. It can give permissions to a service for accessing all resources on a second service, or use a more fine-grained policy in which he associates a list of resources to be accessed for any given service. This possibility, allows permissions to be managed on a centralized level by authorized parties while developer's responsibility is reduced to calling the coordinator before each access.

A second important remark is that this flow is systematic. This allows implementing libraries for managing the authorization on specific development technologies. In this way, the development of RESTful services can be greatly simplified and reduced to declaratively importing a library. The usage of libraries further diminishes the security risks related to incorrect implementations of the above protocol.

#### 4. EVALUATION AND RESULTS

As we mentioned previously, our model is implemented in university area, and as cases there are taken three university applications/services: administrator management system, learning management system (LMS), and e-library.

##### 4.1. Coordinator implementation

The role of our proposed coordinator is to maintain the application keys and generate tokens as required. The coordinator will register a list of actions that each service provides. Given an endpoint, it provides a list of possible URLs, with allowed HTTP verbs for each of them. The combination of the URL (pattern) and the verb may be used as a description of services and for specifying simple permissions.

These tokens will be used by a client application to directly access different services. Services will verify the authenticity of tokens through the coordinator. They may also check on the coordinator whether the client has the necessary permissions to access a given action. The entity relationship diagram (ERD) shown in Figure 3.

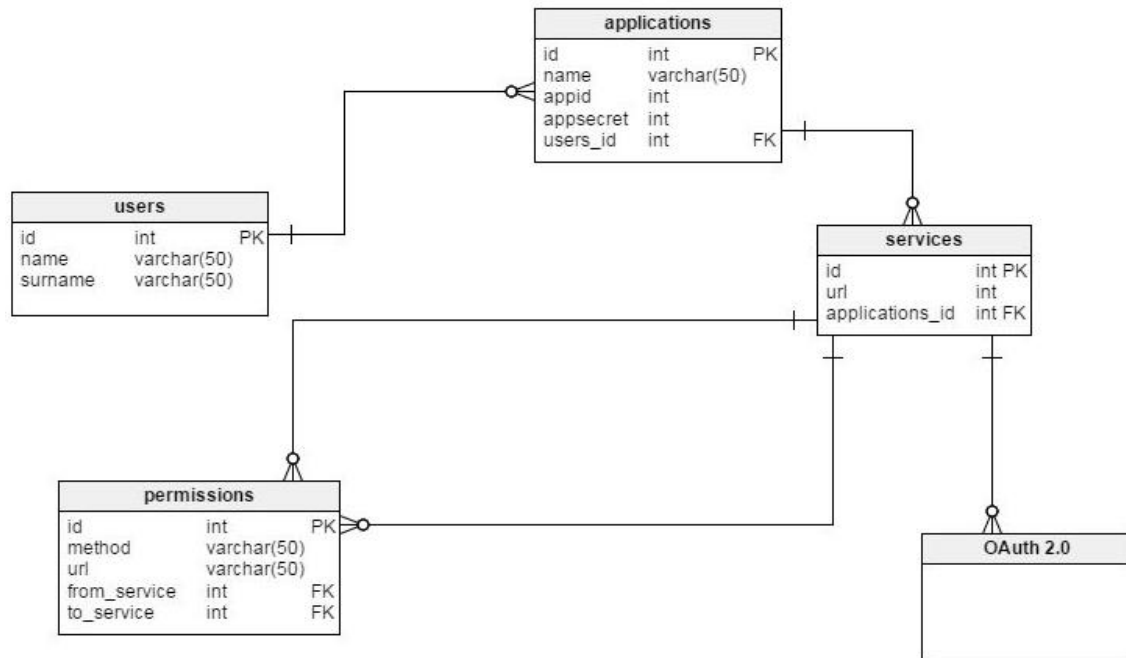


Figure 3. Coordinator ERD

#### 4.2. Requests and REST implementation

In Table 1, service requests are presented with their methods, collection, and operation and business operations based on the proposed framework.

Table 1. Service requests

	From_service	To_service	HTTP method	Collection	Operation	Business operation
1.	A	B	GET	programs	retrieve	Get list of study programs
2.	A	B	POST	faculties	create	Create new faculty
3.	A	B	GET	faculties	retrieve	Get list of faculties
4.	C	B	GET	books	retrieve	Get list of books
5.	A	B	GET	students	retrieve	Get list of students

As defined in the proposed framework, firstly, the application request with functions is presented (request 1 and 3) with the function *insertcourses()*.

```

public function insertCourses()
{
    $result = AdminModel::getdirections();
    $faculties = RestService::requestServiceData('http://lms.dev', 'http://lms.dev/faculties');
    $programs = RestService::requestServiceData('http://lms.dev',
    'http://lms.dev/faculties/1/programs');
    return View::make('admin/insert-courses')->with([
        "data" => json_encode($result),
        "faculties" => $faculties,
        "programs" => $programs,
    ]);
}

```

Then the application makes request to the coordinator with URL.

```
<?php
class RestService {

protected static $appCliendID = "ca0543153c75b8da6a9dcd7c43539df2c5bf3d01";
public static function requestServiceData($service, $requestUrl) {
$token = Session::get('access_token');
    if ($token === NULL) {
        $token = self::getAccessToken();
    }
    if ($token) {
$client = new \GuzzleHttp\Client();
$response = $client->post('http://coordinator.dev/authenticator/request-service', [
        'form_params' => [
            'appID' => $appCliendID,
            'service' => $service,
            'request_url' => $requestUrl,
            'access_token' => $token
        ]
    ]);

        return json_decode($response->getBody()->getContents());
    }

public static function getAccessToken() {
$client = new \GuzzleHttp\Client();
$response = $client->post('http://coordinator.dev/oauth/access_token', [
        'form_params' => [
            'grant_type' => 'client_credentials',
            'client_id' => 'ca0543153c75b8da6a9dcd7c43539df2c5bf3d01',
            'client_secret' => '1ce6299662bf88002804f34e413d82c26be9153e'
        ]
    ]);

$content = json_decode($response->getBody()->getContents());
$token = $content->access_token;
$expires_in = $content->expires_in;
Session::put('access_token', $token);
Session::put('expires_in', $expires_in);
return $token;
    }
}
```

The coordinator controls the request and response to service request.

```
class AuthenticateController extends Controller {

public function construct()
{
    //$this->middleware('oauth', ['only' => 'login']);
}
public function requestService(Request $request) {
$appID = \Input::get('appID');
$serviceId = \Input::get('service');
$requestUrl = \Input::get('request_url');
$data = \Input::get('faculty_name');
$app = Application::where('appid', $appID)->first();
if ($app) {
    $endpoint = $permissions->where('endpoint', $serviceURL)->first();
    $method = $endpoint->method;
    try {
        $response = $this->makeRequest($method, $requestUrl, $data);
        $code = $response->getStatusCode();
        if ($code === 200) {
            return $response->getBody()->getContents();
        } else {
            return response()->json(['message' => 'not allowed', 'code' => '405']);
        }
    } catch (BadResponseException $e) {
        return response()->json(['message' => $e->getMessage(), 'code' => '405']);
    }
    } else {
        return response()->json(['message' => 'No Permissions', 'code' => '405']);
    }
}
```

Figure 4 presents the implemented user interface where the administrator of the system can take rest objects such as study programs and faculties from the other (A) service and add them in the (B) service such LMS. In Figure 5 the implemented user interface is presented, where the administrator of the system such the registered faculties can be saved also in other applications with post method, such from service (A) to service (B).

Figure 4. UI for service response 1 and 3

Next, the function storeRest for the 2<sup>nd</sup> service request is presented.

```
public function storeRest(Request $request)
{
    $facultyName = $request->input('name');
    $response = $this->requestServiceData("http://lms-usht.dev", "http://lms-
    usht.dev/addFaculty", $facultyName);
    return $response;
}
```

Figure 5. UI for 2<sup>nd</sup> service response



Here is presented the function for the 4<sup>th</sup> service request.

```
public function dashboard()
{
    $result = HomeModel::userdata();
    $requestUrl = 'http://elibrary.dev/student/'.Session::get('userid').'/books';
    $serviceData = RestService::requestServiceData('http://elibrary.dev', $requestUrl);
    $hasServiceData = true;
    if ( is_object($serviceData) && $serviceData->code === "405") {
        $hasServiceData = false;
    }
    return View::make('student/dashboard')->with([
        "data" => json_encode($result),
        "serviceData" => $serviceData,
        "hasServiceData" => $hasServiceData
    ]);
}
```

In Figure 6 the implemented User Interface is presented such as the student panel of service (B) or LMS using REST services enables students to see their borrowed books from service (C) or e-library. In Figure 7 the implemented user interface is presented, such as, that the LMS professor panel which enables professors to see their registered students to specific course using REST architecture and our implemented framework and registered students from service (A) to service (B).

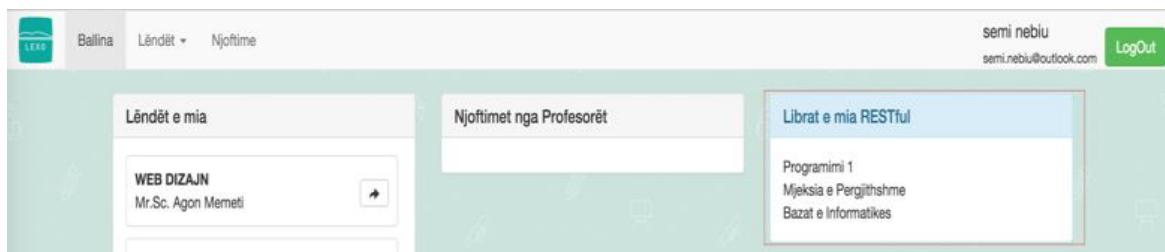
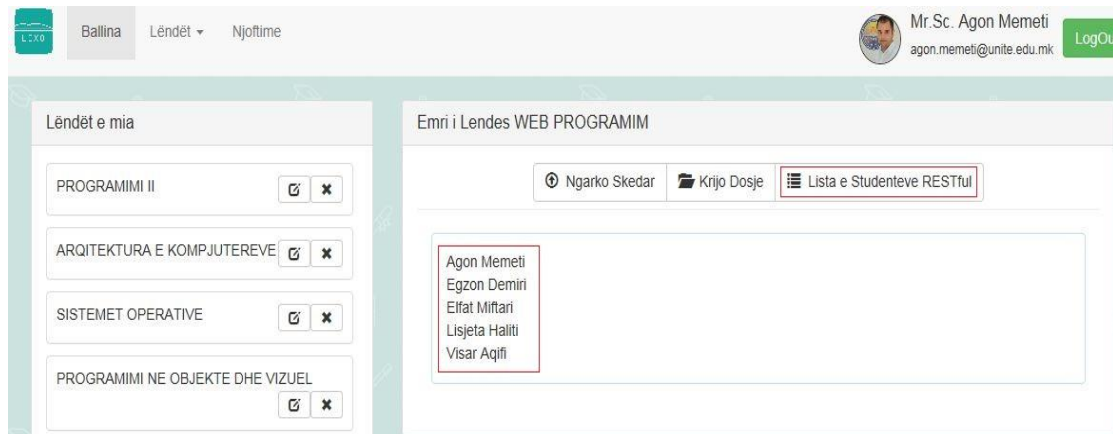


Figure 6. UI for 4<sup>th</sup> service response

Last, the function for the 5<sup>th</sup> service request is presented.

```
public function myCourse()
{
    $c_id = base64_decode(Request::segment(2));
    $validator = Validator::make(array('id' => $c_id), array(
        'id' => 'required|integer'
    ));
    if($validator->passes()){
        $response = ProfessorModel::getCourseData($c_id, Session::get('userid'));
        if($response->result === TRUE){
            $course = Course::find(Request::segment(3));
            $course_service = RestService::requestServiceData('http://lms.dev',
                'http://lms.dev/courses/' . $course->id);
            return View::make('professor/my-course')->with([
                "data" => json_encode(intval($c_id)),
                "course_name" => $course->course_name,
                "course" => $course_service
            ]);
        }else{
            print_r($response->msg);die();
        }
    }else{
        print_r("BAD URL");die();
    }
}
```

Figure 7. UI for 5<sup>th</sup> service response

## 5. CONCLUSION

The suggested prototype promoted in this paper aimed at managing the technological needs of universities effectively, especially integrating all services in a unique platform and having stable services. The purpose was to demonstrate how to better integrate them, especially the interoperability issues between services concerning their integration reusing existing services and changing one service by not attacking the others. Using it in educational institutions will promote huge benefits such as high available services, scalability, increasing portability, performance/cost and overall reliability improvements. As REST services are a challenging task; improving the way that university services are implemented and maintained as well, in this work we have described our proposed framework to use existing services for decoupling authorization for reduced service dependency. Decoupling authorization from the actual implementation of web services simplifies their development while allowing centralized management of permissions by third party authorities. This allows for more flexibility in service implementation and its evolution. Services can be developed and run independently, even when centralized management of permissions is a must. The implemented coordinator handles aspects of registration and authorization of services and he also provides libraries for developers to handle authorization concerns.

In this paper we present architecture (protocol) for in-house REST service coordinator for service collaboration, by exporting data as basic services (resources) through SOA approach. Using the defined protocol, resources are shared, based on the requirements, as what this paper is trying to resolve and propose. A large number of services are designed in a manner to offer closed and isolated system interface and resources due to their specific requirements they need to offer to the users. Later on, isolation consequences are considered, taking University services as a typical in-house organization that dispose many data, where students and faculty staff members need to have more freedom and access to different service resources. This is done for developers to build new services and for students and staff to achieve their course activities, duties and services that the faculty might require to apply and organize. The SOA approach, especially REST services, offer flexible service solutions, first of all by sharing the computational resources based on the service criteria's where each service delivers its power value based on the environment and the technology designed. It also develops easier and just in time solutions.




## REFERENCES

- [1] R. Mahowald, "A brief history of saas," 2009. <http://www.computerworld.com/pdfs/Service-Now-BriefhistoryofSaaS.pdf>
- [2] S. Cholia, D. Skinner, and J. Boverhof, "NEWT: A RESTful service for building high performance computing web applications," in *2010 Gateway Computing Environments Workshop (GCE)*, Nov. 2010, pp. 1–11. doi: 10.1109/GCE.2010.5676125.
- [3] N. A. C. Tavares and S. Vale, "A model driven approach for the development of semantic RESTful web services," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services - IIWAS '13*, 2013, pp. 290–299. doi: 10.1145/2539150.2539193.
- [4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2013.
- [5] T. Fredrich, "RESTful service best practices," *REST API Tutorial*, 2012. <http://toddfredrich.com/restful-best-practices-v1-1.html> (accessed Mar. 23, 2015).
- [6] W. Li and P. Svärd, "REST-based SOA application in the cloud: a text correction service case study," in *2010 6th World Congress on Services*, Jul. 2010, pp. 84–90. doi: 10.1109/SERVICES.2010.86.
- [7] E. Hammer, D. Recordon, and D. Hardt, "The oauth 2.0 authorization protocol," 2011. <http://tools.ietf.org/html/draft-ietf-oauth-v2-25> (accessed Dec. 16, 2014).




- [8] I. Claros, R. Cobos, E. Guerra, J. de Lara, A. Pescador, and J. Sanchez-Cuadrado, "Integrating open services for building educational environments," in *2013 IEEE Global Engineering Education Conference (EDUCON)*, Mar. 2013, pp. 1147–1156. doi: 10.1109/EduCon.2013.6530253.
- [9] IETF OAuth Working Group, "OAuth 2.0." <http://oauth.net/2/> (accessed Dec. 12, 2014).
- [10] IBM WebSphere Application sServer, "OAuth 2.0 service provider and TAI," *IBM Corporation*, 2012. [https://mediacenter.ibm.com/media/OAuth+2.0+service+provider+and+TAI/0\\_fgg4teki](https://mediacenter.ibm.com/media/OAuth+2.0+service+provider+and+TAI/0_fgg4teki) (accessed Dec. 16, 2014).
- [11] Ping Identity, "The essential OAuth primer: understanding OAuth for securing cloud APIs," *Ping Identity*, 2011. <https://www.pingidentity.com/en/resources/content-library/white-papers/3119-essential-oauth-primer.html> (accessed Dec. 16, 2014).
- [12] K. Togias and A. Kameas, "An ontology-based representation of the Twitter REST API," in *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, Nov. 2012, pp. 998–1003. doi: 10.1109/ICTAI.2012.85.
- [13] HSC Proprietary, "Securing RESTful web services using spring and OAuth 2.0," *Hughes Systique*, 2011. <https://hsc.com/About-Us/News-Events/Securing-RESTful-Web-Services-Using-Spring-and-OAuth-20> (accessed Dec. 16, 2014).
- [14] J. Meng, S. Mei, and Z. Yan, "RESTful web services: a solution for distributed data integration," in *2009 International Conference on Computational Intelligence and Software Engineering*, Dec. 2009, pp. 1–4. doi: 10.1109/CISE.2009.5365234.
- [15] A. Memeti, B. Selimi, A. Besimi, and B. Cico, "A framework for flexible REST services: decoupling authorization for reduced service dependency," in *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, Jun. 2015, pp. 51–55. doi: 10.1109/MECO.2015.7181861.

## BIOGRAPHIES OF AUTHORS



**Agon Memeti**    received his B.Sc., M.Sc., and Ph.D. degree from SEE University, Faculty of Contemporary Sciences and Technologies. He is currently an Associate Professor of Computer Sciences at University of Tetova–North Macedonia. Presently he is also working as a head of the office for scientific research and innovation. His main research interests focus on software reuse, software development, optimization algorithms, information retrieval, data mining, and text mining. He can be contacted at email: [agon.memeti@unite.edu.mk](mailto:agon.memeti@unite.edu.mk).



**Betim Cico**    received his B.Sc., M.Sc., and Ph.D. degree from Polytechnic University of Tirana, Albania. He is currently a Full time Professor of Computer Engineering at Metropolitan University in Tirana. Currently he is also working in many research projects dealing with artificial intelligence. His main research interests focus on software engineering, software development, optimization algorithms, information retrieval, data mining, and text mining. He can be contacted at email: [betim.cico@gmail.com](mailto:betim.cico@gmail.com).