Publishing and Consuming RESTful Web API Services

Yurii Boreisha, Oksana Myronovych
Department of Computer Science, Minnesota State University Moorhead, Moorhead, MN, USA
Department of Computer Science, North Dakota State University, Fargo, ND, USA

Abstract - This paper is dedicated to the key issues of building RESTful Web API services using ASP.NET and Entity Framework. It shows how to enable cross-origin requests and secure ASP.NET Web API using token-based authentication. Examples of JavaScript clients that consume RESTful Web API services are provided.

Keywords: REST, Web API, cross-origin requests, token-based authentication.

1 Introduction

Contemporary software development is heavily built on publishing and consuming web services. As expected of any software products, these services and their clients should be maintainable, testable, and adaptive to change. Web API technology provides a way to implement a RESTful web service using all of the goods that the .NET framework offers. In Web APIs, communication between applications is done over the underlying HTTP protocol. With Web API we create endpoints that can be accessed using a combination of descriptive URLs and HTTP verbs: GET – Request data/resource, POST – Submit or create data/resource, PUT – Update data/resource, and DELETE – Delete data resource [1-4].

Cross-Origin Resource Sharing (CORS) mechanism is required when a client requests access to a resource (for example, data from an endpoint) from an origin (domain) that is different from the domain where the resource itself originates. In the CORS workflow, before sending a DELETE, PUT, or POST requests, the client sends an OPTIONS request to check that the domain from which the request originates is the same as the server. The server must include various access headers that describe which domains have access: Access-Control-Allow-Headers (ACAH) header describes which HTTP verbs are supported/permitted [5].

Since Web API adoption is increasing at a rapid pace, there is a serious need for implementing security for all types of clients trying to access data from Web API services. It can be done with token-based authentication that allows to authenticate and authorize each request by exposing OAuth2 endpoints using OAuth2 Authorization Framework and Open Web Interface for .NET (OWIN) [6].

This paper is dedicated to the key issues of building RESTful Web API services using ASP.NET and Entity Framework. It shows how to enable cross-origin requests and secure ASP.NET Web API using token-based authentication. Examples of JavaScript clients that consume RESTful Web API services are provided.

2 Creating a RESTful Web API Service

It is easy to create a Web API service with Individual User Accounts using Visual Studio 2017. Let us assume that we would like to use this service to enable CRUD operations for a simple database (Figure 1). To create all data access classes we can add an ADO.NET Entity Data Model (database first approach). To enable CRUD operations we should create a Web API controller with actions, using entity framework (Figure 2).

The RESTful Web API service is ready. We can add Swashbuckle/Swagger UI to represent API operations – just install Swashbuckle using NuGet Package Manager. Swashbuckle seamlessly adds a Swagger to Web API projects. In order to have a direct link to the Swagger API interface one should add an action link to the top navigation (_Layout.cshtml). To publish this services using IIS Express – just start the application (Figure 3).
3 Creating a JavaScript Client

Let us create a simple HTML/CSS/JavaScript client that allows to retrieve, add, delete, update, and find entries in the Phone Book database. Fragments of jQuery code used in the client application are shown in Figures 4-9. For this application to work one should enable CORS (Cross-Origin Resource Sharing) for the service:

- Enable CORS (`App_Start/WebAPIConfig.cs`): `config.EnableCors();`
- Allow CORS for `Controllers/PhoneBooksController.cs`: `[EnableCors(origins:"*", headers:"*", methods:"*";)]`

![Figure 1: Phone Book Database](image1)

![Figure 2: Web API Controller Settings](image2)

![Figure 3: Swagger UI](image3)
```javascript
var uri = 'http://localhost:32992/api/PhoneBooks';

$(document).ready(function() {
    $.ajax({
        type: 'GET',
        url: uri,
        dataType: 'json',
        success: function (data) {
            $('#addressResult').html("Phone book entries: <br \>");
            $.each(data, function (key, item) {
                $('tr').append('<td>' + item.LastName + '</td>' +
                '<td>' + item.FirstName + '</td>' +
                '<td> ' + item.PhoneNumber + '</td></tr>
            });
        }
    }).fail(function (data) { alert(JSON.stringify(data)); })
});
```

Figure 4: jQuery Code

```javascript
function showAll() {
    $.ajax({
        type: 'GET',
        url: uri,
        dataType: 'json',
        success: function (data) {
            $('#addressResult').html("Phone book entries: <br \>");
            $.each(data, function (key, item) {
                $('tr').append('<td>' + item.LastName + '</td>' +
                '<td>' + item.FirstName + '</td>' +
                '<td> ' + item.PhoneNumber + '</td></tr>
            });
        }
    }).fail(function (data) { alert(JSON.stringify(data)); })
}
```

Figure 5: Function showAll()

```javascript
function find() {
    $.ajax({
        type: 'GET',
        url: uri + "/#" + $('#findPN').val(),
        dataType: 'json',
        success: function (data) {
            $('#addressResult').html("Entry found: <br \>") +
            data.PhoneNumber + ", " +
            data.LastName + ", " + data.FirstName;
            $('#phoneNumber').val($.$trim(data.PhoneNumber));
            $('#lastName').val($.$trim(data.LastName));
            $('#firstName').val($.$trim(data.FirstName));
        }
    }).fail(function (data) { alert(JSON.stringify(data)); })
}
```

Figure 6: Function find()
function add() {
  var newAddress = { LastName: $('&#last_name').val(),
    FirstName: $('&#first_name').val(),
    PhoneNumber: $('&#phone_num').val() };

  $.ajax({
    type: "POST",
    data: JSON.stringify(newAddress),
    url: url,
    contentType: "application/json"
  }).done(function(res) {
    $('#address_result').html("Entry added successfully.");
    document.getElementById("lastName").value = "";
    document.getElementById("firstName").value = "";
    document.getElementById("phone_num").value = "";
  }).fail(function (res) {
    $('#address_result').html("You already have this phone number in your book.");
  });
}

function update() {
  var newAddress = { LastName: $('&#last_name').val(),
    FirstName: $('&#first_name').val(),
    PhoneNumber: $('&#phone_num').val() };

  $.ajax({
    type: "PUT",
    data: JSON.stringify(newAddress),
    url: uri + "/" + $('&#phone_num').val(),
    contentType: "application/json"
  }).done(function (res) {
    $('#address_result').html("Entry updated successfully.");
    document.getElementById("lastName").value = "";
    document.getElementById("firstName").value = "";
    document.getElementById("phone_num").value = "";
  }).fail(function (res) {
    $('#address_result').html("You don't have this phone number in your book.");
  });
}

function deleteEntry() {
  $.ajax({
    type: "DELETE",
    url: uri + "/" + $('&#phone_num').val(),
    contentType: "application/json"
  }).done(function (res) {
    $('#address_result').html("Entry deleted successfully.");
    document.getElementById("lastName").value = "";
    document.getElementById("firstName").value = "";
    document.getElementById("phone_num").value = "";
  }).fail(function (res) {
    $('#address_result').html("You don't have this phone number in your book.");
  });
}

Figure 7: Function add()

Figure 8: Function update()

Figure 9: Function delete()
3 Securing Web API using Token-Based Authentication

Created RESTful Web API service with Individual User Accounts (Visual Studio 2017) includes an authorization server that validates user credentials and issues tokens where Web API controllers act as resource servers. An authentication filter validates access tokens, and the [Authorize] attribute is used to protect a resource. The following diagram shows the credential flow in terms of Web API components (Figure 10). To enable the token-based authentication the following changes should be made for the service and related clients.

For the RESTful Web API service we should do the following [5, 6]:

- Enable SSL.
- Add the ‘RequireHttpsAttribute’ filter to the MVC pipeline (App_Start/FilterConfig.cs)
- Add a custom ‘RequireHttpsAttribute’ filter to the Web API pipeline (App_Start/WebAPIConfig.cs)
- Remove ‘AllowInsecureHttp’ from ‘OAuthOptions’ (App_Start/Startup.Auth.cs)
- Allow CORS for Controllers/AccountController.cs and enable CORS for the /Token endpoint adding the proper code to the App_Start/Startup.Auth.cs (ConfigureAuth method).
- Enable authorization for all related controllers – add the [Authorize] attribute.

For the client app we should add the Registration and Login features, and modify all functions to work with the Authorization headers (Figures 11-15).

---

Figure 10: Credential Flow

Figure 11: jQuery Code (Part 1) - Token-Based Authentication

```javascript
var uri = 'https://localhost:44364/api/PhoneBooks';
var uri2 = 'https://localhost:44364/api/Account/Register';
var uri3 = 'https://localhost:44364/Token';

// If we already have a bearer token, set the Authorization header.
var token = sessionStorage.getItem('tokenKey');
var headers = {};
if (token) {
    headers.Authorization = 'Bearer ' + token;
}
```
```javascript
$(document).ready(function () {
  $.ajax({
    type: 'GET',
    url: url,
    headers: headers,
    dataType: 'json',
    success: function (data) {
      $('#addressResult').html("Phone book entries: <br \">");
      $.each(data, function (key, item) {
        $("<tr><td>' + item.LastName + '</td><td>' + 
          item.FirstName + '</td><td>' + 
          item.PhoneNumber + '</td></tr>").appendTo($('#addressResult'))
      });
    }
  }).fail(function (data) { alert(JSON.stringify(data)); });
});
```

Figure 12: jQuery Code (Part 2) - Token-Based Authentication

```javascript
function find() {
  $.ajax({
    type: 'GET',
    url: url + "/" + $('#findFirstName').val(),
    headers: headers,
    dataType: 'json',
    success: function (data) {
      $('#addressResult').html("Entry found: <br \">");
      data.LastName + "", " + data.FirstName); 
      $('#phoneNum').val($('#trim(data.PhoneNumber)'));
      $('#lastName').val($('#trim(data.LastName)'));
      $('#firstName').val($('#trim(data.firstName)'));
    }
  }).fail(function (data) { alert(JSON.stringify(data)); });
}
```

Figure 13: Function find() – Token-Based Authentication

```javascript
function register() {
  var data = {
    Email: "coreisha@mnstate.edu",
    Password: "Password1!",
    ConfirmPassword: "Password1!"
  };

  $.ajax(
    type: 'POST',
    url: url2,
    contentType: 'application/json',
    data: JSON.stringify(data)
  ).done(function(data) {
    $('#addressResult').html("Registration is OK!");
  }).fail(function(data) {
    $('#addressResult').html("Registration failed! You should register only once!");
  });
}
```

Figure 14: Function register() – Token-Based Authentication
4 Conclusion

This paper is dedicated to the key issues of building RESTful Web API services using ASP.NET and Entity Framework. It shows a ‘big picture’ development approach based on the following steps:

- Creating a RESTful Web API service.
- Adding Swashbuckle/Swagger UI to represent API operations/endpoints.
- Sharing Web API using Token-Based Authentication.
- Creating JavaScript/jQuery clients to consume RESTful Web API services.

Future developments will be dedicated to the following topics and related new technologies, protocols, standards, and frameworks:

- What kind of API to use (for the present - RESTful, SOAP, JavaScript, XML-RCP)?
- What data format to use in the request (for the present - JSON, HTML, XML)?
- What kind of authorization/authentication to use (for the present – OAuth2, HTTP Basic Authentication)?

5 References