**Customer profile management service**

**[High level architecture]**

**Version history**

|  |  |  |
| --- | --- | --- |
| Version | Date | Author |
| 1.0.0 | 01/08/2018 | Kalpa Senanayake |

**Introduction**

Customer profile management service will be built as a part of the new project to offer customers an omni-channel experience from our mobile application and web application. It offers real-time profile management to customers.

Traditional CRM services are not capable enough to offer sophisticated interfaces and future challenges brought to the table by these two frontiers.

**Assumptions**

In version 1.0.0 of the service we make following assumptions in order to deliver an outcome form the development effort. If these assumptions are not true readers are welcome to point those out and discuss furthermore.

|  |  |
| --- | --- |
| **Assumption** | **Impact on the development** |
| The CRM is legacy system and do not expose REST API out of the box it provides POX messages over HTTP and currently, current versions of these service endpoints are in level 0 of the Richardson maturity model. | Until CRM team come up with the REST APIs’ the project will complete the development on using these existing services. |
| A unique customer identifier, which is currently used in organization wide, will be used to identify customer in this project as well. | It makes lot of sense and will speed the development. |
| The consumers of these services (both mobile and web clients) are well equipped to consumers of these web services over HTTP using JSON as the Data format and are able to consume JSON. | Allow the development team to use well known architectural and development standards. |
| The mobile app is not using query based REST API invocation like GraphQL or  gRPC or Protocol Buffers. | Allow the development team to use well-known HTTP protocol and RESTful architectural and development standards. |
| These REST API service layer only do the transformation XML to JSON and act as adapter layer to the mobile and web client. No business logic is going to be moved to new layer. | It simplifies the initial versions of the API development. |
| These services are near-real-time synchronous services. | In this version, project wants to keep out the asynchronous capabilities of the profile management as out of scope. |
| Due to security recommendation, these services only operate over SSL. | Additional effort for cost for certificate management. |
| This service deployed to militarized section of the network zone and exposed to internet through an API gateway. | Allow the development team to leverage the benefits of existing API gateway pattern. |

**Service definition**

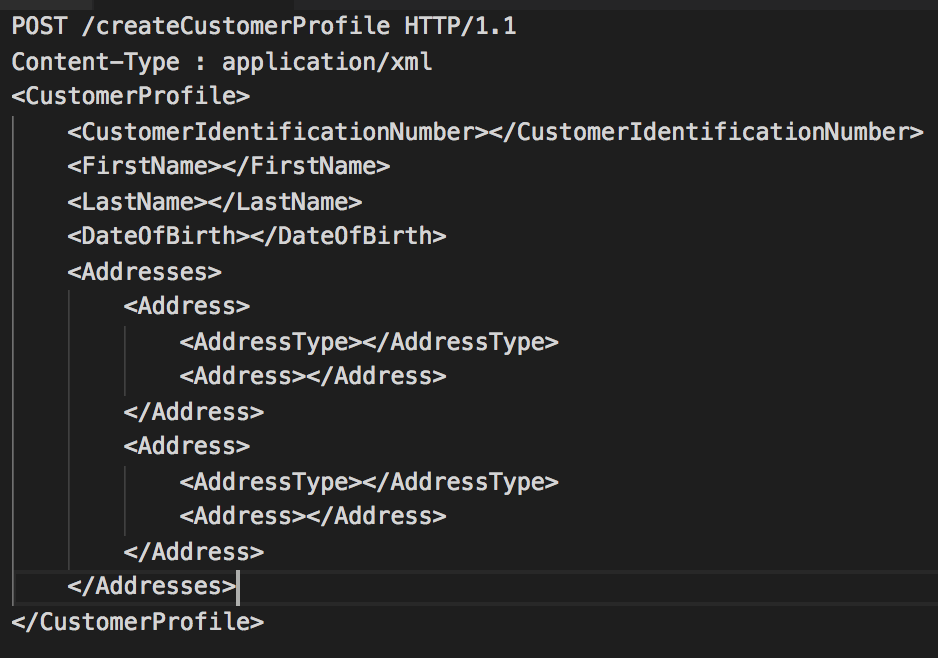
The customer profile management micro service will be bounded to manage customer profile data in CRM system. The business capability that it offers will only be limited to profile management features of both web and mobile clients.

It offers, create, read, update and delete RESTFul resources to manage customer profile.

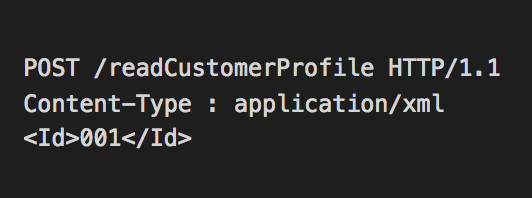
**Characteristics of the existing CRM back end**

It is really important to understand the characteristics of the existing CRM customer profile management endpoints. Following sample request formats will give reader an idea about the request payload format.

#. /createCustomerProfile and /updateCustomerProfile request.



#. /readCustomerProfile and /deleteCustomerProfile request.



**Architecture**

High level integration flow can be explained as it shows in the following diagram. This diagram take **createCustomer** resource as an example to show the characteristics of the integrated system which can be used to understand the rest of the flows.



**Mobile or web client:** Service consumer web application or mobile application.

**APIG :** API gateway.

**Customer service:** Customer service implementation.

**CRM\_POX\_API:** CRM legacy backend with POX data format.

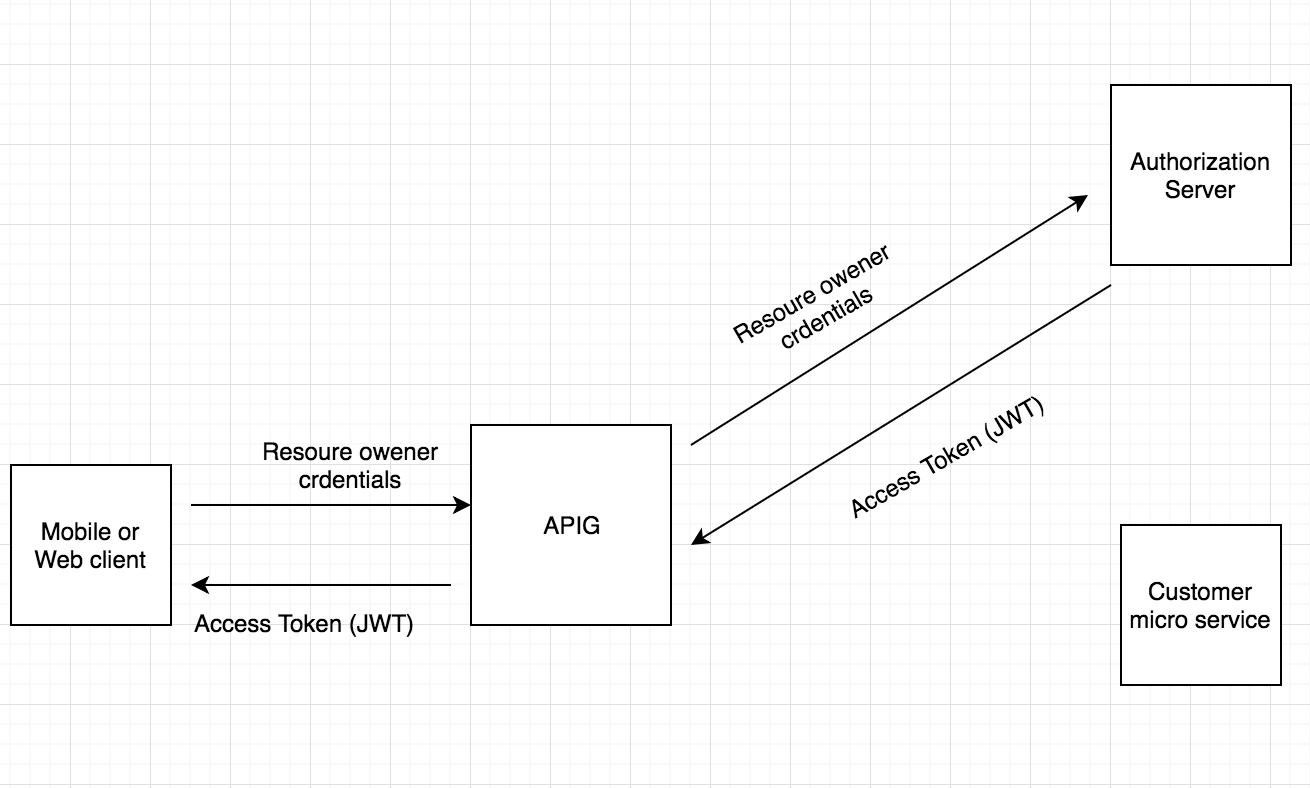
**Architecture with OAuth 2.0 Security**

The diagram in the above section deliberately avoids the security mechanism since this section talks about it specifically. According to the requirement of the project by design these services should be able to serve mobile and web clients, and must be capable of handling authentication and authorization on top of it’s services.

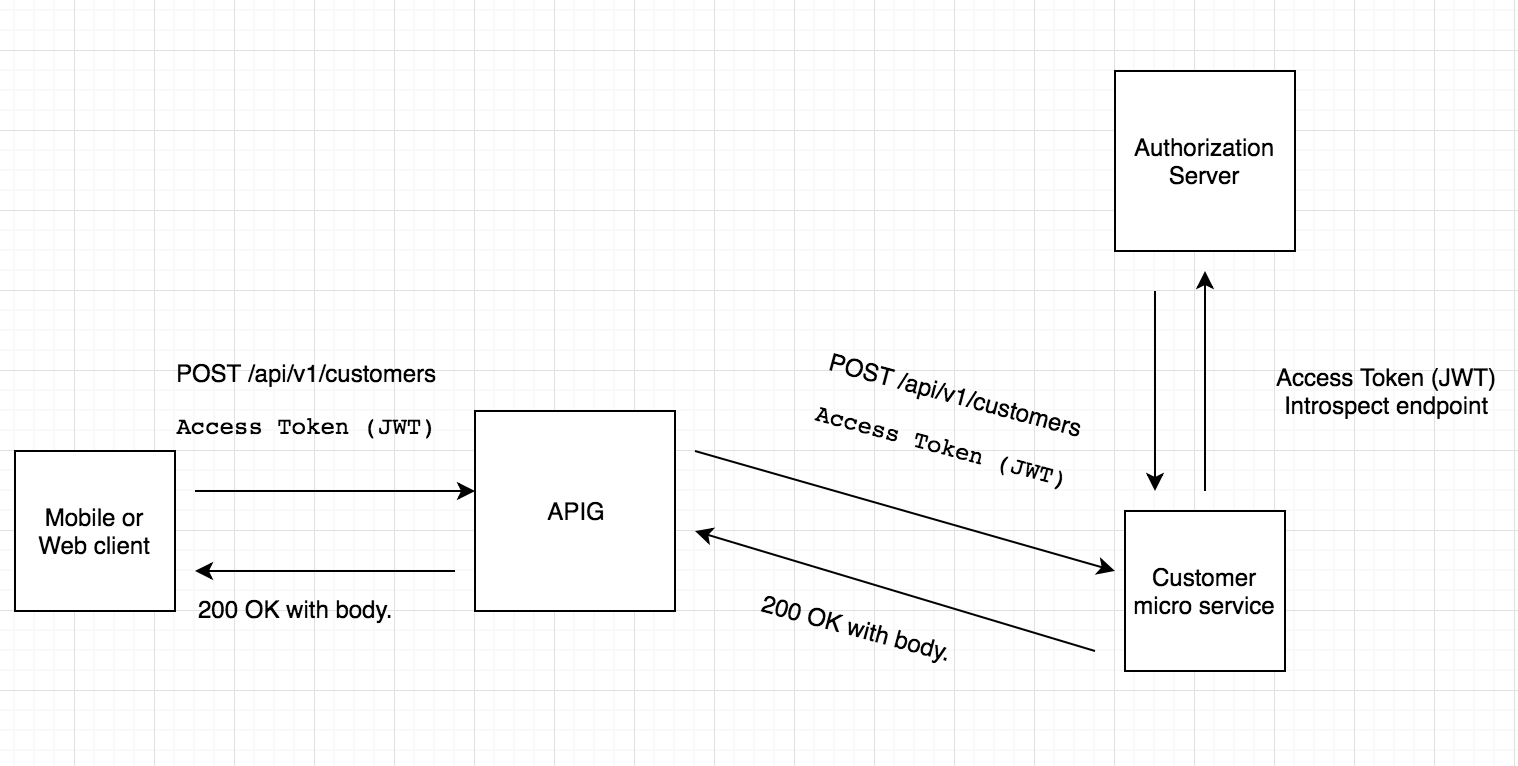
Suggested approach to address this is to use the “Authorization delegation pattern”, which delegates the authentication and authorization to an internal authorization server.

In this pattern, the authorization server takes care of the credential validation and issues a JSON Web Token with scoped set of privileges. And the client uses this JWT to access service. The following diagrams explain how it works in two steps.

Step 1: Acquiring an access token from authorization server.



Step 2: Utilizing the access token to access the customer service.



Since this solution should support for both mobile and web clients service, it will leverage OAuth 2.0 specification’s two grant types to acquire access token.

**Access token for web client.**

Web client will use “Authorization Code” grant type. In the first step client acquires an authorization code, which the client use to exchange for an access\_token and a refresh\_token.

The refresh\_token can be used to get a new access\_token when it expires.

**Access token for mobile client.**

Mobile client will use “Resource Owner Credential” grant type. This grant type client acquires an access\_token by authenticating user using customer credentials against authorization server.

**Implementation**

For the implementation of above solution, first preference is to use the existing internal Identity and access manager as the authorization server. The “introspect” endpoint will allow individual customer service instances to query for access token validity. This approach allows microservices to manage their service level security by it’s own.

If organization does not have any existing access manager, the project will have to implement an OAuth 2.0 compatible token management service. One possible implementation can be done using spring security via “WebSecurityConfigurerAdapter” and “AuthorizationServerConfigurerAdapter”.