

**IMC- Tax calculator Vendor integration**

Architecture Design and Technical Specification



Version: *Draft*

Table of Contents

[1 Purpose 2](#_Toc57534413)

[2 Synopsis 2](#_Toc57534414)

[3 Conceptual Architecture and Design 2](#_Toc57534415)

[3.1 Sequence Diagram 3](#_Toc57534416)

[3.2 Class Diagram 4](#_Toc57534417)

[4 Major Architecture Qualities and Principles 4](#_Toc57534418)

[4.1 Architecture Reusability 4](#_Toc57534419)

[4.2 Extensibility 4](#_Toc57534420)

[4.3 Scalability 4](#_Toc57534421)

[4.4 Security 4](#_Toc57534422)

[4.5 Availability and Reliability 5](#_Toc57534423)

[4.6 Notifications and Alerts 5](#_Toc57534424)

[4.7 Instrumentations 5](#_Toc57534425)

[4.7.1 Logging and Tracing 5](#_Toc57534426)

[5 IMC TaxService Overview 5](#_Toc57534427)

[5.1 Retrieving Tax Rate By Location 5](#_Toc57534428)

[5.2 Calculate Tax For a given Order 5](#_Toc57534429)

[6 Source Control Repository and CICD 6](#_Toc57534430)

[7 Tax Calculator Vendor Information 6](#_Toc57534431)

[7.1 Taxjar 6](#_Toc57534432)

[8 Testing the APIs 7](#_Toc57534433)

### 1 Purpose

The purpose of this document is to detail the Architecture Design decisions that will be used to implement a solution for IMC and Tax Calculator Vendor integration. This document explains the API Client Integration implementation plan with Taxjar APIs and provides details to accommodate further integrations to add additional vendor APIs or replace the Taxjar with another vendor.

This document is intended to be used as a reference during the analysis and design phase and as an implementation guide in development and integration phases. The audiences for this document are the project stakeholders, project managers, business analysts, application architects and solution developers.

### 2 Synopsis

IMC uses a lot of external services and APIs to accommodate the customers’ needs. One of them is Tax calculation. There are lot of Tax calculation APIs out there and IMC needs to be able to work with many of them via a common interface. This solution is to create a Tax Service that integrates with the third-party Tax calculator and provide functionality to return the total tax that needs to be collected.

The initial implementation is based on a third-party vendor ‘Taxjar’ as Tax Calculator. This solution provides the functionality to consume the Taxjar Tax Calculator as a client. The solution is implemented keeping in mind for further new integrations into the IMC TaxService and ability to replace Taxjar with another third-party vendor.

The implementation provides the following two functions:

* Get the Tax rates for a location
* Calculate the taxes for an order

### 3 Conceptual Architecture and Design



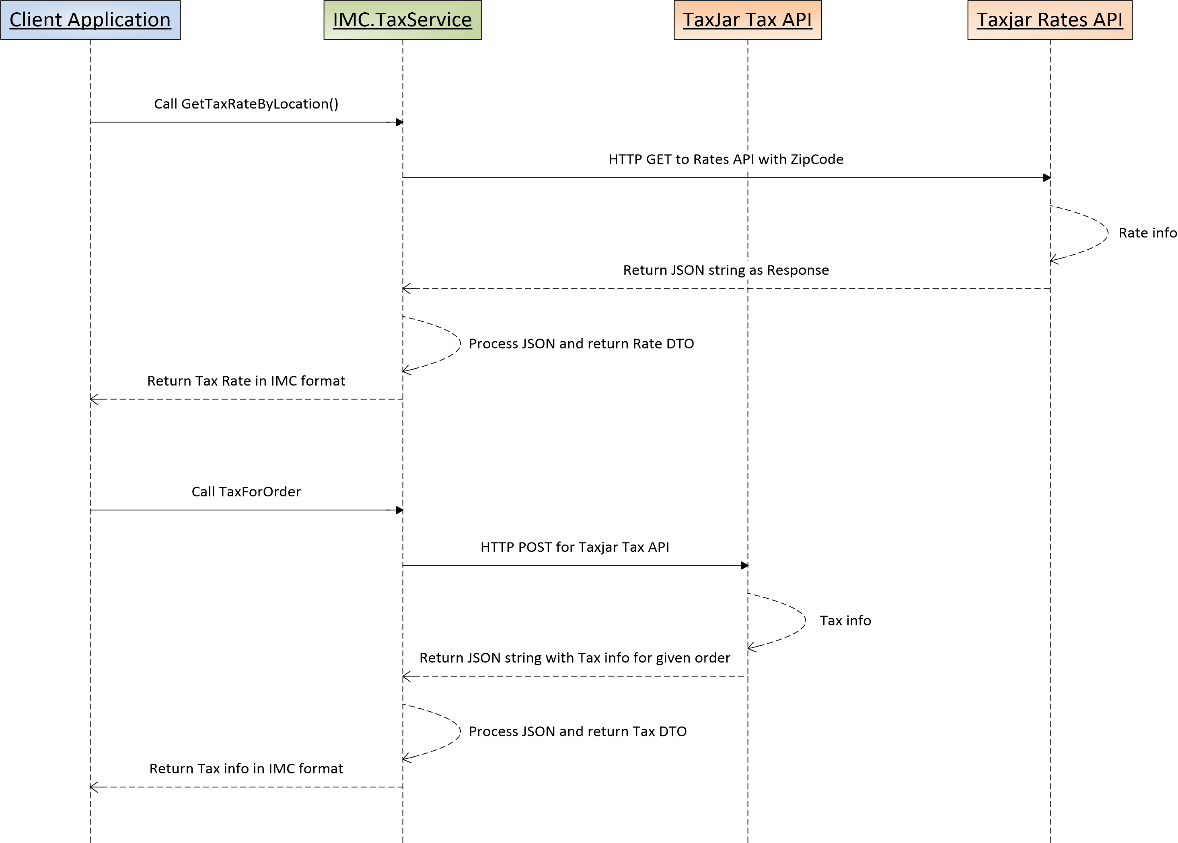
* **Tax Calculator Vendor:** ‘Taxjar’ vendor APIs are being consumed for Rates and Tax information.
* **API Client / Service Agent (TaxjarService)**: This Service provides the functionality to make HTTP GET and POST calls to ‘Taxjar’ APIs. Response JSON string will be parsed into ‘Taxjar’ Entities and further to IMC standard DTOs. Returns Tax and Rate DTOs as response. This service is injected as a dependency in IMC TaxServiceController using the default IoC Dependency Injection container.
* **Vendor Entities:** These objects are defined in ‘TaxjarService’ for the JSON string transformation to objects. These objects further used to transform into IMC standard DTOs. These vendor entities may change depending on the Tax Calculator vendor.
* **DTOs:** Considering these as IMC Standard Objects to be consistent with all the vendor integrations. Data Contracts to consume the client data retrieved from the Vendor APIs. Database schema will be depending on these objects schema.
* **TaxService Controller:** Contains the core implantation of the External API Client ‘TaxjarService’ integration. IMC TaxService APIs for Rates and Taxes are exposed as RESTful APIs for other IMC applications and clients.
* **Components and Routing:** Routing, External API Client Services and other components like configuration access and logging are initialized using Dependency Injection.

**Swagger** is used for describing the structure of the APIs and to test/validate , immediately right in the browser, that the implemented APIs work as they should and to allow for seamless integration with automated API testing in future.

**AutoMapper** is used define an object-object mapping strategy. In this solution we use it to map Vendor Entities to IMC DTOs and vice versa.

* **TaxService APIs:** This IMC TaxService is the face of the entry point to get the TaxRate for a Location and Tax for a given Order Info. The internal integration with Third party vendor ‘Taxjar’ will be encapsulated to IMC clients.
* Generic Helper Components to process the HTTP requests and manage the JSON strings for business object conversion.

#### 3.1 Sequence Diagram



#### 3.2 Class Diagram



### 4 Major Architecture Qualities and Principles

Below are some architectural qualities that will be driving the design and development of the solution.

#### 4.1 Architecture Reusability

The solution should leverage, to the extent possible, any existing IMC enterprise application artifacts to reduce development time.

#### 4.2 Extensibility

The recommended solution architecture should leverage the principles set forth for modern and multi-platform framework. Embrace dependency injection by leveraging the ASP.NET Core built-in dependency injection container which provides the necessary hooks and infrastructure to inject dependencies.

#### 4.3 Scalability

The Solution Services should be designed with scalability in mind to support increasing load with minimal impact to the application. Consider Service Fabric and keep the APIs simple, micro, modular and accommodate future growth.

#### 4.4 Security

All service calls must be properly authenticated and authorized. APIs/Services implementation should adhere to the Security specifications and policies enforced by IMC leadership team.

#### 4.5 Availability and Reliability

Services should be resilient enough to be able to recover from total hardware failure within the time frame specified in the service level agreement(s) as implemented by the business. Consider multiple Regions and Availability Zones deployment in Azure with auto scaling policies enabled.

#### 4.6 Notifications and Alerts

Solution may consider the Azure Alerts and Monitoring metrics to keep track of the health status of the Services. Services should leverage the internal mechanisms to manage for Application Logic specific alerts and error notifications.

#### 4.7 Instrumentations

##### 4.7.1 Logging and Tracing

All errors must be logged in a common and accessible location with the error, user, server, time, method and parameters being recorded. Error Logging should always be on. Consider Azure Application Insights for Track Tracing.

### 5 IMC TaxService Overview

#### 5.1 Retrieving Tax Rate by Location

This method from IMC TaxService is used to retrieve the Rate information from Vendor Tax Calculator API for a given ZipCode of a location. This method returns serializable data contract ‘Rate’. As this is exposed as a RESTful WEB API internal IMC Applications or external clients can consume with a simple HTTP GET.

**Method Name**: GetTaxRateByLocation

**End Point:** <<BaseURL>>/api/tax/taxrates/{ZipCode}

**Request Parameters**: ZipCode as integer

**Response Object**: Rate (Please see the Class diagram above for Rate object properties)

**Implementation:** This method consumes the External API Client Service ‘TaxjarService’ API method ‘GetTaxRateByLocaton’ with the following Request data and receives Rate object as response.

* **APIRequestData:**

APIKey :{Configuration value of ‘Taxjar: APIKey’ from appSettings.json}

BaseUrl: {Configuration value of ‘Taxjar:BaseUrl’ from appSettings.json}

* **ZipCode**  - a 5 digit integer value

APIRequestData is a generic request object for all API calls that are initiated from IMC TaxService

#### 5.2 Calculate Tax For a given Order

This method from IMC TaxService is used to retrieve the Tax information from Vendor Tax Calculator API for Order information. This method returns serializable data contract ‘Tax’. As this is exposed as a RESTful WEB API internal IMC Applications or external clients can consume with a simple HTTP POST.

**Method Name**: GetTaxRateByLocation

**End Point:** <<BaseURL>>/api/tax/taxfororder

**Request Parameters**: OrderData (Please see class diagram above for OrderData object properties)

**Response Object**: Rate (Please see the Class diagram above for Rate object properties)

**Implementation:** This method consumes the External API Client Service ‘TaxjarService’ API method ‘TaxForOrder’ with the following Request data and receives ‘Tax’ object as response.

* **APIRequestData:**

APIKey :{Configuration value of ‘Taxjar: APIKey’ from appSettings.json}

BaseUrl: {Configuration value of ‘Taxjar:BaseUrl’ from appSettings.json}

* **OrderData**  - order information with line items. Please see the above class diagram for all the properties.

APIRequestData is a generic request object for all API calls that are initiated from IMC TaxService

### 6 Source Control Repository and CICD

### 7 Tax Calculator Vendor Information

#### 7.1 Taxjar

We are only going to be talking to their SalesTax API:  
[https://developers.taxjar.com/api/reference/#sales-tax-api](https://urldefense.com/v3/__https:/developers.taxjar.com/api/reference/*sales-tax-api__;Iw!!AGfBXFdQIw!EizFgfyghVmNU7m7uqBpM3gMrGpl12PjvkvYDNkzs4Qtuln4aRLFKl-AVQ-8$)

Here is the API Key:  
5da2f821eee4035db4771edab942a4cc

**API Endpoint for Rate for Location:**

[https://api.taxjar.com/v2/rates/{ZipCode}](https://api.taxjar.com/v2/rates/%7bZipCode%7d) (HTTP GET)

**Test Data for ZipCode:** 92093

**API Endpoint for Tax for Order:**

<https://api.taxjar.com/v2/taxes> (HTTP POST)

**Test Data for Request ‘Order’ Info:**

{

"from\_country" : "US",

  "from\_zip" : "92093",

  "from\_state" : "CA",

  "from\_city" : "La Jolla",

  "from\_street" : "9500 Gilman Drive",

  "to\_country" : "US",

  "to\_zip" : "90002",

  "to\_state" :"CA",

  "to\_city" : "Los Angeles",

  "to\_street" : "1335 E 103rd St",

  "amount" : 15,

  "shipping" : 1.5

}

### 8 Testing the APIs

**Postman:**

We may start with the postman testing to make sure the vendor provided APIKey and URLs are working. A simple HTTP GET / HTTP POST calls for the APIs will confirm the connectivity and provide us the JSON objects to start our class schema design.

You may use the included postman collection for testing:



**Swagger:**

Integrate the Swagger in the solution by adding the NuGet package and initiate the swagger services in the Service Startup class.

services.AddSwaggerGen();

Change the launch URL to swagger, so that you can test the services with the swagger default api test page.

Swagger also provides information about the Data contracts associated with the API methods implementation.

**Unit Tests:**

We have 2 Unit test project associated with the solution.

* **IMC TaxService Unit Test**

This unit test project provides 2 Test methods to test Rates and TaxForOrder APIs of the local Tax Service.

* **Taxjar APIs Unit Test**

This unit test project provides 2 Test methods to test Rates and TaxForOrder APIs directly with Taxjar interfaces