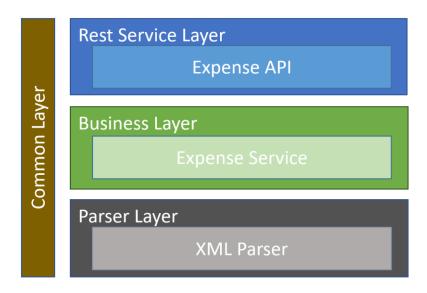
Serko Expense API

1. High Level Architecture



The application is implemented using layered architecture. Each layer adhering to the SOLID principles. Following contains a high level description of each layer.

Rest Service Layer (SerkoExpense.Api):

- Contains the Web API endpoint to import data.
- App_Start\WebAPIConfig.cs
 - Register the attribute routing, log4net configuration, ExceptionFilter and Autofac configuration.
 - Autofac is the DI framework used to inject dependencies. GetAutofacContainer method register the controller and other types and return the container. This is used to set the dependency resolver.
- Filters\ExceptionFilter.cs
 - Handled exceptions using custom exception attribute because then all the exceptions will be handled at single point and it can be logged.
 - This filter logs the exception using log4net and throw custom exception message to user. Log4net configuration is under log4net section in web.config
 - The filter registered in WebApiConfig.
- Controllers\ ExpenseController.cs
 - o Contains the ImportData API endpoint method to import text content.
 - o This is a POST method because data should be sent on request body.
 - Accepts the string textContent as input and returns ImportResponse object

- CORS enabled using the web.config settings.
 - o Current settings allow any origin, any domain and any method.
 - Used web.config settings instead of configuring in WebApiConfig because it will allow to change CORS settings without rebuild and redeploy

Business Layer (SerkoExpense.Business):

- Contains the business service implementation.
- IService.cs
 - This is the service interface
- ExpenseService.cs
 - o This is the service implementation. This class implements the IService.
 - This method calculates the GST and total excluding GST.
 - The GST percentage is configured under app settings (key= GST) in Web.config of Web API project.

Parser Layer (SerkoExpense.XmlParser):

- Contains the parser which will validate the input string and extract the content.
- IParser.cs
 - This is the parser interface
- XmlParser.cs
 - o This contains the XML parser implementation. The class implements the IParser.
- ReservationDTO cs
 - o This is the DTO class which keeps and transfer the deserialized XML content
- ParserResponse.cs
 - This class transfer the parser status and ReservationDTO

Common Layer (SerkoExpense.Common):

- Contains the common classes that is shared among all other layers.
- Constants.cs
 - o This class contains the constant values used across the application
- Enums.cs
 - This class contains the enumerations used across the application
- Expense.cs
 - This class contains the expense information extracted from XML and calculated data (GST and total excluding GST)
- Reservation.cs
 - This class contains the reservation information including expense
- ImportResponse.cs
 - This is the response object returned from the API. It contains the response status, message and reservation information.

2. Development environment

.Net framework: 4.6.1

IDE: Visual Studio 2017 Community Edition

3. API endpoint, request and response

Endpoint Url: http://localhost:51214/Expense/ImportData

Request: string

- This is the text received via email
- Send string in JSON format

Response: ImportResponse

- This contains three properties
 - Success- Indicates whether import success or not
 - Message- Contains the error message if import unsuccessful. Will be null if import successful.
 - Reservation-Contains the extracted and calculated data

Sample request/response from Postman:

```
http://localhost:51214/Expense/ImportData
    POST ▼
                                                                                                                                                                                                                              Send
                                                                                                                                                                                                                                                      Save •
                                                            Body • Pre-request Script
             Authorization Headers (1)
"Hi Yvaine,
          Please create an expense claim for the below. Relevant details are marked up as
          requested...
<expense><cost_centre>DEV002</cost_centre>
         <total>1024.01</total><payment_method>personal card</payment_method>
         //expense>
From: Ivan Castle
Sent: Friday, 16 February 2018 10:32 AM
To: Antoine Lloyd <Antoine.Lloyd@example.com>
         Subject: test
        Subject: test 
Hi Antoine, 
Please create a reservation at the <vendor>Viaduct Steakhouse</vendor> our 
<description>development team's project end celebration dinner</description> on 
<date>Tuesday 27 April 2017</date>. We expect to arrive around 
7.15pm. Approximately 12 people but 1'11 confirm exact numbers closer to the day. 
Regards, 
Tyan
17 Ivan
18 "
                                                                                                                                                                      Status: 200 OK Time: 46 ms Size: 801 B Save Download
Body Cookies Headers (13) Test Results
             Raw Preview JSON ▼ 👼
                                                                                                                                                                                                                                                         ■ Q
  Pretty
1 + {
                  "Success": true,
"Message": null,
                 "Message": null,
"Reservation": {

"Vendor": "Viaduct Steakhouse",
"Description": "development team's project end celebration dinner",
"Date": "Tuesday 27 April 2017",
"Expense": {

"CostCenter": "DEV002",

"Total": 1024.01,
"PaymentMethod": "personal card",
"GST": 153.6015,
"TotalScaluningST": 970 4005
                              "TotalExcludingGST": 870.4085
```

4. Testing

Unit test project (SerkoExpense.Api.Test) is done to test the Expense API. Test methods available to test following scenarios.

No	Scenario	Test Method	Status
1	Opening tags that have no corresponding closing tag. In this case the whole message should be rejected.	<pre>ImportData_ShouldRejectIfCorrespondingClosingTagIsMissing</pre>	Pass
2	Missing <total>. In this case the whole message should be rejected.</total>	ImportData_ShouldRejectIfTotalTagIsMissing	Pass
3	Missing <cost_centre>. In this case the 'cost centre' field in the output should be defaulted to 'UNKNOWN'.</cost_centre>	ImportData_ShouldReciveUnknowForEmptyCostCenter	Pass
4	extracted and calculated data should be available when xml is valid	ImportData_ShouldReturnExtractedAndCalculatedDataWhenXMLIsValied	Pass

5. Assumptions

- 1. Value of <total> can be zero
- 2. Value of <total> cannot be negative
- 3. <total> cannot have empty string as value (<total> </total> is not allowed)
- 4. <total> cannot be empty tag (<total> </total> is not allowed)
- 5. Self-closing tags will not consider as valid, therefore will be ignored.