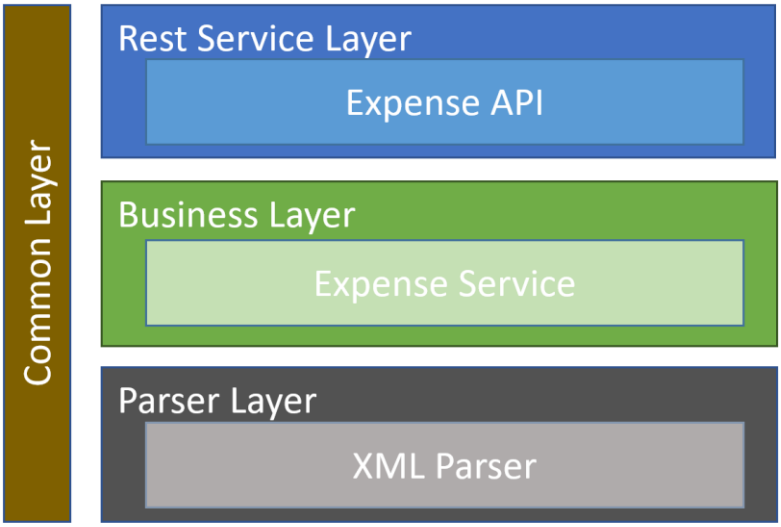
Serko Expense API

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
    - Contains the ImportData API endpoint method to import text content.
    - 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.
    - 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
    - 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
    - This contains the XML parser implementation. The class implements the IParser.
- ReservationDTO.cs
    - 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
    - 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

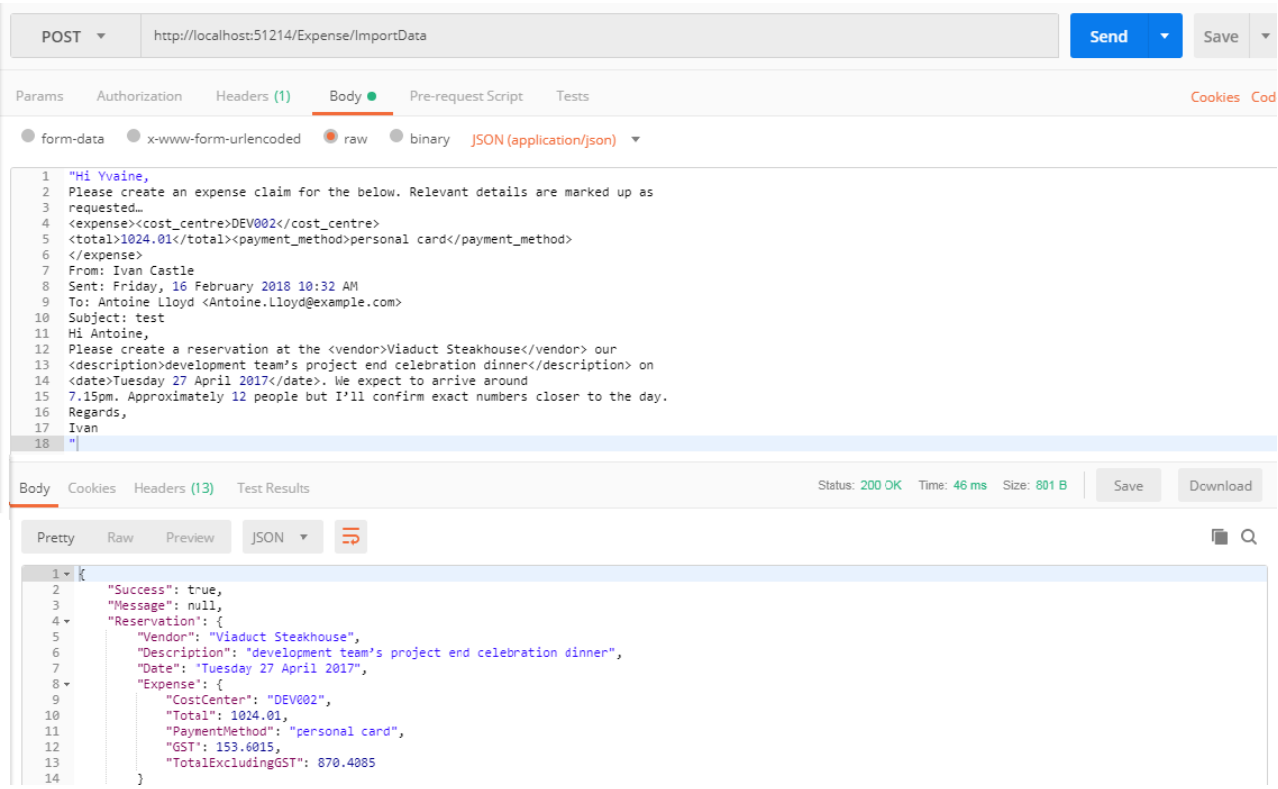
Endpoint Url: http://localhost:51214/Expense/ImportData
Request: string

- 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:



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.	`ImportData_ShouldRejectIfCorrespondingClosingTagIsMissing`	Pass
2	Missing `<total>`. In this case the whole message should be rejected.	ImportData_ShouldRejectIfTotalTagIsMissing	Pass
3	Missing `<cost_centre>`. In this case the 'cost centre' field in the output should be defaulted to 'UNKNOWN'.	`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.