System Design Document (SDD)

01/01/2017

*Babel APIs []*

Document Control Section

Version Number: V.01

Authors

|  |  |  |
| --- | --- | --- |
| Role | Name | Date |
| Head of Software Development | Richard King | 06/01/2017 |

Reviewers

|  |  |  |
| --- | --- | --- |
| Role | Name | Date |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Version No. | Change By | Change Date | Change Description |
| V.01 | Narender D | 01/01/2017 | Initial version |

**Table of Contents**

[1. Introduction 1](#_Toc515013008)

[1.1. Scope 1](#_Toc515013009)

[1.2. Overview 1](#_Toc515013010)

[2. Application Overview 2](#_Toc515013011)

[2.1. General Description 2](#_Toc515013012)

[3. High Level Component Architecture 3](#_Toc515013013)

[4. Application overview 4](#_Toc515013014)

[4.1. Technology Choices 4](#_Toc515013015)

[4.1.1. .NET Core 4](#_Toc515013016)

[4.1.2. Dapper 4](#_Toc515013017)

[4.1.3. Swashbuckle/Swagger/OpenAPI 4](#_Toc515013018)

[4.1.4. F# 4](#_Toc515013019)

[4.1.5. Beatpulse 4](#_Toc515013020)

[4.1.6. Fluent Validation 4](#_Toc515013021)

[4.1.7. Identity Server 4](#_Toc515013022)

[4.1.8. sERILOG 4](#_Toc515013023)

[4.1.9. SIGNALR 4](#_Toc515013024)

[4.1.10. XUNIT 4](#_Toc515013025)

[4.1.11. FAKEITEASY 4](#_Toc515013026)

[4.1.12. MOQ 4](#_Toc515013027)

[4.1.13. GIRAFFE 4](#_Toc515013028)

[4.2. Application organisation 5](#_Toc515013029)

[4.2.1. Folder structure 5](#_Toc515013030)

[4.2.2. Feature folders 5](#_Toc515013031)

[4.2.3. CommonApiComponents 5](#_Toc515013032)

[4.2.4. Private vs Corporate segregation 5](#_Toc515013033)

[4.3. Security Architecture 5](#_Toc515013034)

[4.3.1. Security considerations 5](#_Toc515013035)

[4.3.2. HMAC 5](#_Toc515013036)

[4.3.3. Owasp mitigations 5](#_Toc515013037)

[4.3.4. Access control (sso/tokens) 5](#_Toc515013038)

[4.4. Patterns and principles 5](#_Toc515013039)

[4.4.1. General principles (12 factors inc. Intialistion, Healthchecks) 5](#_Toc515013040)

[4.4.2. Caching 5](#_Toc515013041)

[4.4.3. Result Class & Exception Handling 5](#_Toc515013042)

[4.4.4. Controllers/Handlers/Query | Command | ServiceClient 5](#_Toc515013043)

[4.4.5. Async 5](#_Toc515013044)

[4.4.6. Middleware 5](#_Toc515013045)

[4.5. Non Functional Attributes 5](#_Toc515013046)

[4.5.1. Load and scale 5](#_Toc515013047)

[4.5.2. Auditability 5](#_Toc515013048)

[4.6. Monitoring and fault management 5](#_Toc515013049)

[4.6.1. Logging 5](#_Toc515013050)

[4.6.2. Correlation ids 5](#_Toc515013051)

[4.6.3. Instrumentation/Analytics 5](#_Toc515013052)

[4.7. Deployment CI/CD pipeline 6](#_Toc515013053)

[4.7.1. Phoenix environments 6](#_Toc515013054)

[4.7.2. Team city build processes 6](#_Toc515013055)

[4.7.3. Deployment stages and purposes 6](#_Toc515013056)

[4.8. Environments 6](#_Toc515013057)

[5. Development methodology/process 7](#_Toc515013058)

[6. Testing 8](#_Toc515013059)

[6.1. Types 8](#_Toc515013060)

[6.1.1. Ownership of tests and where and when they are run in the SDLC 8](#_Toc515013061)

[6.1.2. Unit/Integration tests 8](#_Toc515013062)

[6.1.3. PACT testing 8](#_Toc515013063)

[6.1.4. Post deployment testing (Postman collections) 8](#_Toc515013064)

[6.1.5. QA regression testing 8](#_Toc515013065)

[6.1.6. UAT 8](#_Toc515013066)

[6.1.7. Nonfunctional : Performance testing 8](#_Toc515013067)

[7. Application Setup for Local Development Environment 9](#_Toc515013068)

[7.1. Insert instructions on how to set up local dev environment 9](#_Toc515013069)

[7.2. Unit testing set up 9](#_Toc515013070)

[8. Troubleshooting guide 10](#_Toc515013071)

[9. Know Issues 11](#_Toc515013072)

[10. Reference Documents 12](#_Toc515013073)

# Introduction

An SDD is a record of architectural design, solution structure and development and testing process. It will also include functional and technical information about the system, its architecture and its dependencies. The audience for an SDD is broad and covers development, QA, support and DevSecOps.

## Scope

The scope of this SDD targets the web application tier of the applications that comprise the Babel project.

## Overview

The SDD will:

* Describe technology and architectural choices
* Describe solution structure and organization.
* Include details of build and deployment pipeline systems and processes
* Include reference to development methodology
* Describe security considerations.
* Describe approach to testing at all layers of the application.
* Include reference to NFRs

# Application Overview

## General Description

Babel is Moneycorp’s Foreign Exchange (FX) and International Payments platform

It is an externally accessible application that allows private and corporate (Beta) customers access to a sub-set of Omni functionality.

Babel is a user based application, functionality and privileges are based on user’s type. In the first login user needs to provide GDPR preferences.

**For Private customers:**

For private customers, the foreign exchange is limited to FX Spot and must be paid for at the point of booking the deal. Pricing comes from Reuters, and is saved to the Moneycorp database and actually booked via FX-All via the Auto-Booker. A payment wizard guides them through the process and captures required information.

The user can add funds, Sell currency (dealing), Add Recipients, Send Payments, Track a rate, view their statement, and Apply/Manage their Explorer Card.

**For Corporate customers:**

For corporate customers, the foreign exchange can be either a spot deal or a forward (up to 2 years in the future). Corporate users are not required to pay for their deal at the point of booking. But they have the option to pay with funds on account or direct debit. Deals are booked directly with FxAll.

The user can add Fund; both Buy and Sell currency, Set Value date, Add Recipient, Send Payment. Track Rate, Account Statement. Get Live Rate from FxAll rate service.

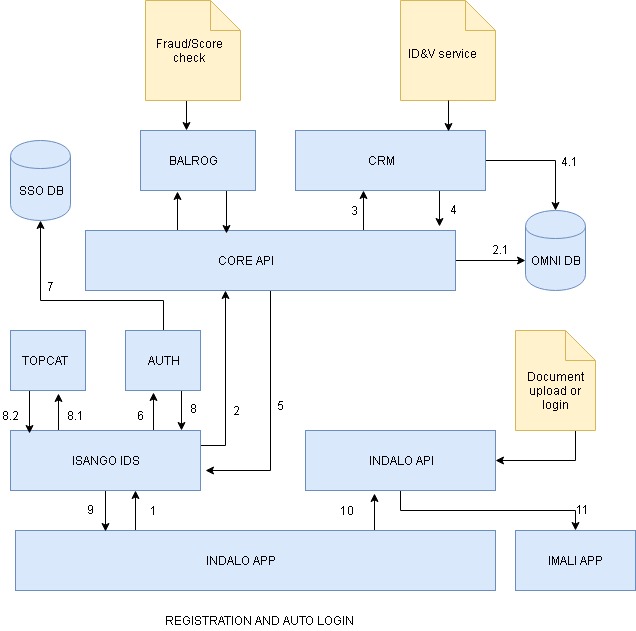
A prospective customer can open accounts using the website, subject to successfully passing necessary Anti-Money Laundering/Fraud checks.

# High Level Component Architecture

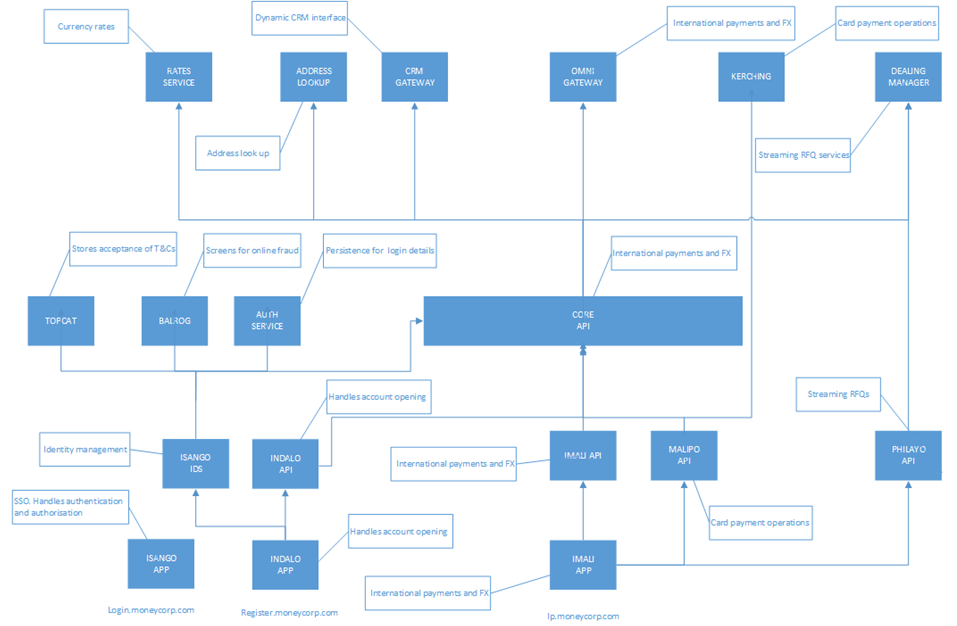
Microservices Architecture design:

## 

**Registration Flow Diagram:**



## 



***The highlights****:*

Core: The API façade to the legacy moneycorp systems such as OMNI Gateway, CRM gateway etc

Isango – Gate: Login and Identity Management API and Front-end

Indalo – Creation: Account opening API and Front-end

Imali – Money: International Payments and FX API and Front-end

Malipo – Payment: Card Payment Operations API

Philayo – Living/live: Rate streaming API for Corporate customers.

* *Backend for Frontend:*

An API specifically for a particular Frontend E.g. Imali will make numerous calls to downstream services and aggregate the data into a payload for a specific front end view. Which means the browser isn’t making those numerous calls, it only makes one

* *‘Strangler’ pattern:*

Strategy for breaking apart the monolithic Gateway

* *Mock services:*

Allow easy development, especially for Front-end devs by not requiring the full tech stack

# Application overview

## Technology Choices

### .NET Core

One of the main limitations of the .NET Framework was that it failed to share code across platforms.

.NET Core is a new cross-platform and fully opens source .NET implementation that was forked from .NET Framework and Silverlight. .NET Core provides developers with a library which can be deployed over various platforms and which also allows developers to import just the parts of the Framework they need for their projects.

* **Flexible deployment**: Can be included in your app or installed side-by-side user- or machine-wide.
* **Cross-platform**: Runs on Windows, macOS and Linux; can be ported to other OSes. The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.
* **Command-line tools**: All product scenarios can be exercised at the command-line.
* **Compatible**: .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library.
* **Open source**: The .NET Core platform is open source, using MIT and Apache 2 licenses. Documentation is licensed under CC-BY. .NET Core is a .NET Foundation project.
* **Supported by Microsoft**: .NET Core is supported by Microsoft, per .NET Core Support

### Dapper

Dapper is the second fastest ORM or micro ORM or simply object mapper that maps your database directly with C# objects. [Official Website](http://dapper-tutorial.net/)

The following are the key features of Dapper:

* Speed and fast in performance.
* Fewer lines of code.
* Object Mapper.
* Static Object Binding.
* Dynamic Object Binding.
* Easy Handling of SQL Query.
* Easy Handling of Stored Procedure.
* Operating directly to IDBConnection class that provides smoothness and running query directly to the database instead of passing data using various objects as we do in EF and ADO.NET.
* Multiple Query Support.
* Support for Stored Procedure.
* Bulk Data insert functionality.
* Dapper also allows fetching multiple data based on multiple inputs.

### Swashbuckle/Swagger/OpenAPI

OpenAPI is a specification for describing REST APIs. With OpenAPI, instead of XML, you have set of JSON objects, with a specific schema that defines their naming, order, and contents.

OpenAPI Specification file allow you to describe an API including (among other things):

* General information about the API
* Available paths (/resources)
* Available operations on each path (get /resources)
* Input/Output for each operation

**Short Introduction**:

* OpenAPI = Specification
* Swagger = Tools for implementing the specification

### F#

F# is a strongly typed, multi-paradigm programming language that encompasses functional, imperative, and object-oriented programming methods. F# is most often used as a cross-platform Common Language Infrastructure (CLI) language. [Big Topic]

Used in below applications:

* Cyfeiriad (address loopup)
* Gwirio (IDnV)

### Beatpulse

BeatPulse is a health check library for ASP.NET Core Applications. Used in almost all API to check if all dependencies are up and running.

BeatPlus is Very helpful for infrastructure team to find if anything is breaking.

[NuGet Peckage Info](https://www.nuget.org/packages/BeatPulse/)

[Read more about BeatPulse](https://github.com/xabaril/BeatPulse)

### Fluent Validation

Fluent Validation is an open source validation library for .NET. It supports a fluent API, and leverages lambda expressions to build validation rules. Fluent Validation contains .NET libraries and the validation is performed using the Lambda expression. Use Fluent Validation when you want to create some advanced and complex validation for the user data.

Below are the some advantages:

* Replace traditional Data Annotations validation
* Chaining Validation Rules
* Custom Property Validators

Example:

public CardPaymentInRequestValidator()

{

RuleFor(c => c.CardNumberEnding).Length(4).WithMessage(ErrorCode.InvalidValue.Code());

RuleFor(c => c.ClientId).GreaterThan(0).WithMessage(ErrorCode.NoValue.Code());

}

### Identity Server

IdentityServer is a framework and a hostable component that allows implementing single sign-on and access control for modern web applications and APIs using protocols like OpenID Connect and OAuth2. It supports a wide range of clients like mobile, web, SPAs and desktop applications and is extensible to allow integration in new and existing architectures. We are using IdentityServer4 for .net core.

IdentityServer4 incorporates all the protocol implementations and extensibility points needed to integrate token-based authentication, single-sign-on and API access control in your applications.

### Serilog

Serilog is a diagnostic logging library for .NET applications. It is easy to set up, has a clean API, and runs on all recent .NET platforms. While it's useful even in the simplest applications, Serilog's support for structured logging shines when instrumenting complex, distributed, and asynchronous applications and systems.

[Official Website](https://serilog.net/)

[Serilog vs NLog](https://www.darylcumbo.net/serilog-vs-nlog-benchmarks/)

### Signalr

[Wiki] SignalR is a software library for Microsoft ASP.NET that allows server code to send asynchronous notifications to client-side web applications. The library includes server-side and client-side JavaScript components. [Big Topic]

<https://www.asp.net/signalr>

### Xunit

XUnit.net is a free, open source, community-focused unit testing tool for the .NET Framework. It is written by the original inventor of NUnit v2.

XUnit.net is the latest technology for unit testing C#, F#,and other .NET languages. XUnit.net works with ReSharper, CodeRush, TestDriven.NET and Xamarin.

Other popular unit testing frameworks

* Built-in Visual Studio testing tools
* Ms Test
* NUnit
* XUnit

[XUnit vs NUnit](https://xunit.github.io/docs/comparisons)

### Fakeiteasy

A .Net dynamic, open source fake framework for creating all types of fake objects, mocks, stubs etc.

* Easier semantics, all fake objects are just that - fakes - the use of the fakes determines whether they're mocks or stubs.
* Context-aware fluent interface guides the developer.
* Designed for ease of use.
* Full compatibility with both C# and VB.Net.

[nsubstitute-vs-moq-vs-fakeiteasy](http://blog.netronica.io/nsubstitute-vs-moq-vs-fakeiteasy/)

### MOQ

Moq is a mocking framework for C#/.NET. It is used in unit testing to isolate your class under test from its dependencies and ensure that the proper methods on the dependent objects are being called.

Other mocking frameworks (for .NET) include TypeMock, RhinoMocks, nMock, .etc.

Functionally there are no major differences between Moq and FakeItEasy, the syntax is an obvious different.

### Giraffe

Giraffe is an F# micro web framework for building rich web applications.

We can install the Giraffe from NuGet package or from command *PM> Install-Package Giraffe*.

Giraffe is aimed at F# developers. Babel stack have 2 application which are using F sharp and Giraffe:

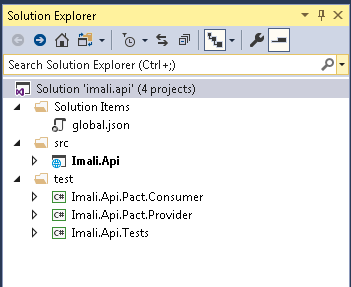
* + - * + gwirio
        + cyfeiriad

## Application organisation

### Folder structure

All Babel API’s follow .net core standard folder structure; the project root folder has 3 main folders:

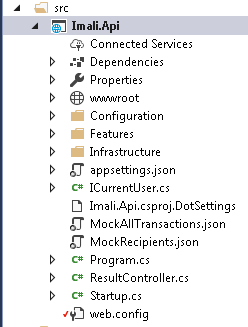
1. Solution Items
2. Src
3. Test



4.2.1.1 **Solutions Items folder**

Solution item folder usually (not limited to) contains application level settings. This folder contains one file global.json, in the file; the project's setting is significant. It tells ASP.NET where to look for your source code and what folders contain your projects. There are two possible folders “src” for source and a “test” folder. Unless your projects and source code are in one of these two folders, the code won't be available to build. You can change these settings if you want.

4.2.1.2 **src folder**



**SRC** is our main project folder. It has below folders.

* Configuration: All configurations related settings are here, we have appSettings and webConfig which contains settings all deployment environments setting.
* Features: All main logic files here – handlers and controllers.
* Infrastructure – API End points and network related settings/initlation/configuration.

### CommonApiComponents

CommonApiConmonents is a useful library which contains common components for ASP.Net Core API projects. CommonApiConmonents is the heart of Babel and is used by most API applications.

This library contains all common logic.

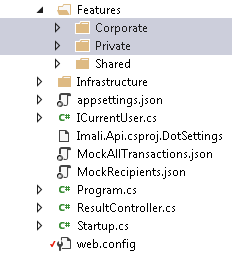
It does below task commonly (not limited to):

* caching
* Security
* Middleware
* Pact testing configuration
* Helper methods
* Exception handling
* Validation
* HMAC.

### Private vs corporate segregation

On the sql server side private and corp segration is done by using clienttype, client type are defined in **ClientType** table. Every client is associated to one type.

In the API(s) ../src/features/ folder have seprate folder for each type which contains logic for respective users.



## Security Architecture

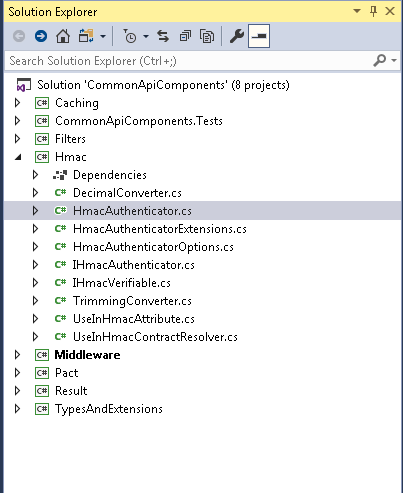
### Security considerations

Token based authentication (Open Id Connect) is used for authentication. Every single request will require the token. The token, is sent on every request in the Authorization header. There is no session based information to manipulate since we don’t have a session.

[Click here to know more on how to use token based authentication.](https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication)

### HMAC

A message authentication code (MAC) is produced from a message and a secret key by a HMAC algorithm. An important property of a HMAC is that it is impossible to produce the HMAC of a message and a secret key without knowing the secret key. A HMAC of the same message produced by a different key looks unrelated. Even knowing the HMAC of other messages does not help in computing the HMAC of a new message.



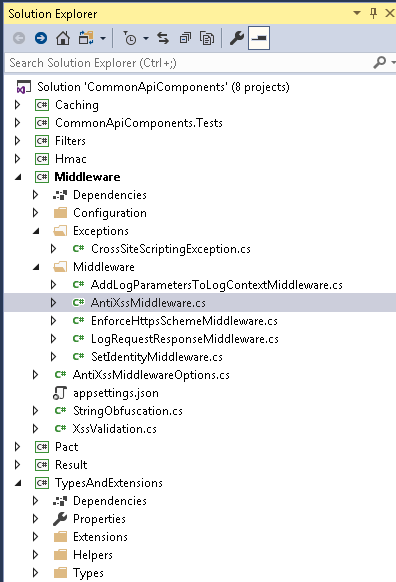
### OWASP mitigations

The Open Web Application Security Project (OWASP) is a 501c3 non for profit educational charity dedicated to enabling organizations to design, develop, acquire, operate, and maintain secure software. All OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security.

The goal of the OWASP Top 10 Proactive Controls project is to raise awareness about application security by describing the most important areas of concern those software developers must be aware of. The top 10 changes periodically but some of the most common vulnerabilities include:

* + Preventing Injection Attacks.
  + Building a proper authentication and session management.
  + Preventing cross-site scripting.
  + Preventing insecure direct object references.
  + Basic security configuration guide.
  + Protecting sensitive data.
  + Ensuring function level access control.
  + Preventing CSRF.
  + Where to look for vulnerabilities on known third-party components.
  + Redirect validation.

[OWASP Top 10 Threats and Mitigations](https://www.owasp.org/index.php/OWASP_Top_10_Threats_and_Mitigations_Exam_-_Single_Select)



### Access control (SSO/tokens)

Single Sign-on authentication creates a centralized login system. SSO is a session and user authentication service that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications.

In Babel, Isango-Ids API is responsible for SSO and token based authentication. Isango-Ids is new sso for Babel. Bable is using **IdentityServer4** for single sign-on.

<https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/>

## Patterns and principles

### General principles (12 factors inc. Initialization, Health checks)

Babel is hoping to implement some of the 12 factors app methodology.

*12 factors are:*

I *Codebase*: There should be exactly one codebase for a deployed service with the code base being used for many deployments.

II *Dependencies*: All dependencies should be declared, with no implicit reliance on system tools or libraries.

III *Config*: Configuration that varies between deployments should be stored in the environment.

IV *Backing services*: All backing services are treated as attached resources and attached and detached by the execution environment.

V *Build, release, run*: The delivery pipeline should strictly consist of build, release, run.

VI *Processes*: Applications should be deployed as one or more stateless processes with persisted data stored on a backing service.

VII *Port binding*: Self-contained services should make themselves available to other services by specified ports.

VIII *Concurrency*: Concurrency is advocated by scaling individual processes.

IX *Disposability*: Fast startup and shutdown are advocated for a more robust and resilient system.

X *Dev/Prod parity* : All environments should be as similar as possible.

XI *Logs*: Applications should produce logs as event streams and leave the execution environment to aggregate.

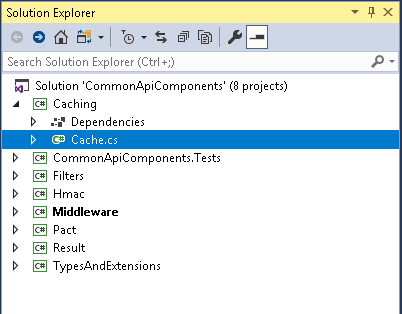
XII *Admin Processes*: Any needed admin tasks should be kept in source control and packaged with the application.

We also have added some other factors such as Health monitoring

*BeatPlus*is implemented Health Checks of applications.

### Caching

Babel have own mechanism for caching. Babel is using Microsoft cache memory. Babel is caching data on load which has less frequency of change. Like a list of countries or RFT values.



Core.Api implementation:

services.AddSingleton(o => new CountryCache(o.GetService<ICountryService>().GetCountries, o.GetService<IMemoryCache>()));

### Result Class & Exception Handling

Exception handle filter is implemented to handle at one place. Using try catch block at some places as well.

CoreApiComponents application is used to handle exception globally.

### Controllers/Handlers/Query | Command | ServiceClient

Controllers are the first interaction point for any communication to the Babel API, the handlers are attached to every action, for every action method called handler execute the request.

Based on the task handler process to request i.e. either call to other API or call other service points or execute the database command.

### Async

Asynchronous execution is implemented in Babel for responsiveness. And we expect to have async methods from the entry point (in the controller action method) to the bottom layer (typically calling another service or querying a database)

### Middleware

Middleware is software that's assembled into an application pipeline to handle requests and responses. Each component:

* Chooses whether to pass the request to the next component in the pipeline.
* Can perform work before and after the next component in the pipeline is invoked.

Babel is using Middleware for (not limited to):

* HMAC
* Log
* CrossSiteScriptingException
* AntiXss

[Click to know more on Middleware](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-2.1&tabs=aspnetcore2x)

## Non Functional Attributes

### Load and scale

**//TODO**

<<See Tarun for Jmeter testing>>

The Babel microservices architecture means we can scale critical apis such as Imali and Core independently to less critical services such as Topcat. All APIs are hosted in a server farm and will be installed on two servers that are load balanced via the F5

### Auditability

All work that we do should be covered by a ticket in Jira

Also, every commit to Git should contain a Jira ticket number

This means the applications can be audited to find out why any given piece of functionality exists and when and why it was added

## Monitoring and fault management

### Logging

Serilog has been used to capture system errors and exceptions. This includes application validations failures and system errors. [Graylog](https://qa-applogs.qamoneycorp.local/streams) is used to access these logs from interactive console.

### Correlation ids

Babel is using correlation ID for Grouping log message and network request. It is used for trace log messages and application flow.

X-Correlation-Id header is used to include correlation IDs in HTTP requests.

### Instrumentation/Analytics

We have a tool called AppDynamics which can give us data for monitoring and analytics. This is yet to be set up on Babel, but may well be at some point in the future

We are also looking at other tools such as Kibana to get better statistics and dashboards

## Deployment CI/CD pipeline

### Phoenix environments

* On demand environments
* Could be used for branch testing
* More information can be found at : <http://jira.moneycorp.local:8090/display/DEVOPS/Phoenix>
* Babel deployment environment to check some special branch functionality, if required we can deploy required branch to test new functionality or feature.

### Team city build processes

* Open [appTeamCity](https://appteamcity.moneycorp.local/project.html?projectId=Babel).
* Select Babel project from the project menu.
* Select the environment on which you want to deploy build example (azure QA, Azure Dev or Azure QZ).
* Select the project we wish to deploy and run the build.
* After build completed, note down the created package.

Now, open [octopus](https://appoctopus/app#/).

* Create new release and select the package (one from TeamCity).

Note: We can skip components which we don't wish to deploy in octopus.

* Select environment and deploy.

### Deployment stages and purposes

Click here for  [Pre-Prod and Prod Environments](http://jira.moneycorp.local:8090/display/BAB/Pre-Prod+and+Prod+Environments) and [Applications/environments matrix](http://jira:8090/pages/editpage.action?pageId=27623876)

*Azure:* <http://jira.moneycorp.local:8090/display/BAB/Azure+Environments>

*QA / Prod:* <http://jira.moneycorp.local:8090/display/BAB/Pre-Prod+and+Prod+Environments>

#### Azure Dev,

* CI deployment tests pass
* Once a day after hours

This is Babel environment for developers.

#### Azure QA,

* Stable Azure Dev
* Deployed automatically

Babel is not using this environment for now. This is QA environment for Omni and other applications.

#### Azure QZ,

* Early indication from master
* Release tag from release branch

This is Babel real QA environment, QA team test from here.

#### QA|QB,

Babel is not using this environment for now. This is QA environment for Omni and other applications.

#### Prod

Babel production environment, we have different server for different applications.

## Environments

| **Environment** | **Application Servers** | **Components** |
| --- | --- | --- |
| [Environment Info](http://jira:8090/pages/editpage.action?pageId=27623876) | Click on link to check AS | Click on link to check |

# Development methodology/process

Babel is developed using agile development methodology. Daily meeting proceeds irrespective of whether all team members are present. Usually the feature team, product owner and scrum master all attends. Usually the meeting takes place around the wall boards. Each team member gives a brief, relevant account of work achieved yesterday, work scheduled for today and more importantly any blocks impeding progress toward the sprint goal.

*Requirements are categorized into 4 types.*

* Epic -Top level container for stories
* User Story -New feature
* Bug -Defect in existing feature
* Spike –Investigation

*Below are the step followed while developing:*

* Requirement gathering
* Architecture and design
* Test planning
* Implementation/Coding
* Testing/QA
* Deployment and maintenance

# Testing

## Types

### Ownership of tests and where and when they are run in the SDLC

The following diagram shows where each type of test is run:

<http://jira.moneycorp.local:8090/display/DEVOPS/Test+Coverage+Matrix>

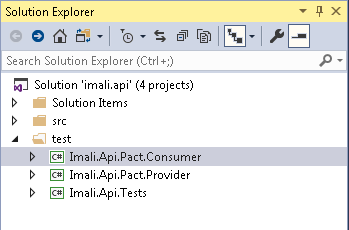
### Unit/Integration tests

Xunit is used for Integration testing and FakeItEasy is used for mock. We can use ReSharper or visual studio to debug test.

### PACT testing

Pact is a contract testing tool. Contract testing is a way to ensure that services (such as an API provider and a client) can communicate with each other.

Pact is a tool which allows you to define an API contract as a json file. This can then be used to test that both the provider and the consumer conform to this contract and identify any breaking changes.



Set up and basic info here.

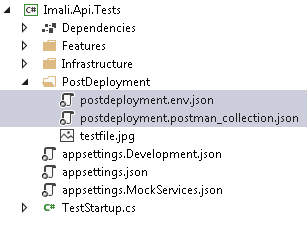
<http://jira.moneycorp.local:8090/display/PACT/Pact+Home>

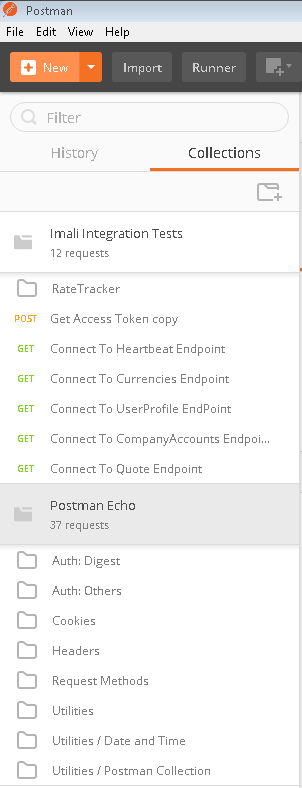
<http://jira.moneycorp.local:8090/pages/viewpage.action?pageId=25625000>

### Post deployment testing (Postman collections)

Every API have automated post-deployment test is implemented to check nothing break after deployment. The test configuration is specified in the postdeployment.env.Json file. postdeployment.postman\_collection.Json have some tests, we can import them to Postman and run or we can make own test cases.

We can use it to test any environment (local or dev or QA) by altering environment URL. Post deployment test can be found in every api(s) **PostDeployment** **folder**.





### QA regression testing

N/A

### UAT

N/A

### Nonfunctional: Performance testing

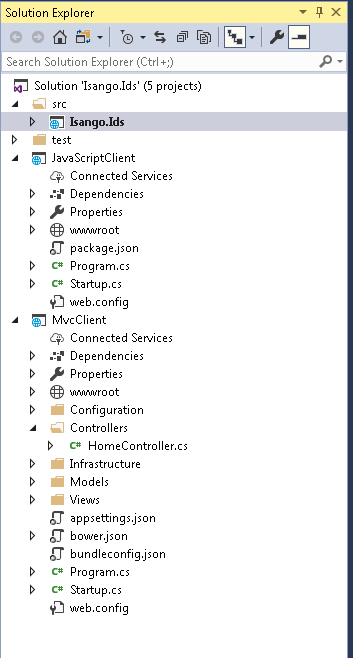
Load / Stress testing (still TODO)

# Application Setup for Local Development Environment

## Insert instructions on how to set up local dev environment

**For front end setup follow frontend setup guide document <link here>**

1. Download and install Dot net core from [Here](https://www.microsoft.com/net/download/windows)
2. Make sure you have access to all Babel applications. (Isango-app, Isango.api,Imali-app,Imali.api,Core.api,Indalo.api,Indalo-app…)
3. Install Git Client on your machine [SETUP GUIDE HERE](http://jira:8090/pages/viewpage.action?spaceKey=TOOL&title=Git+client+installation)
4. Clone all Babel GIT repositories on local using Git or visual studio.
5. Open solution in visual studio 2017 with **<your login>** (adm or non-adm account). In WNS we use adm-account. It should look like below example of Isango.ids



1. Run Application.



Babel API can be run using above steps.

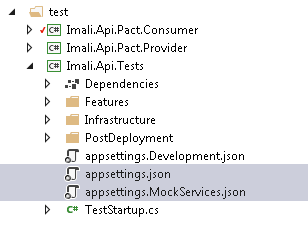
If you want to run all babel stack in once [Use PoweShellScripts](https://appgithub.moneycorp.com/catherallc/PowershellScripts/blob/master/README.md) , a very nice console/terminal emulator [Download Here](http://cmder.net/) to run PowerShellScripts.

## Unit testing set up

Babel is using Xunit.net for unit testing. We can use ReSharper or Visual Studio to run the unit test.

The unit test configuration settings can be found in the appsettings.Json file under test project.

We don’t need special setting to run/debug unit tests.



# Troubleshooting guide

There are various ways Babel troubleshooting can be done (based on bug/error we are challenging). Hereinafter are some ways which could be meaningful for initiate investigation:

1. Serilog can be followed to investigate production issue, using Correlation Ids we can investigate flow and identify which API is yielding an error?

2. For API troubleshooting, post-deployment test suits could be effective.

3. Local setup the Babel project and investigates (if required).

4. Connect local setup to QA API(s).

5. Check the database to make sure it is not a data issue. Seldom is the issue because of data inconsistencies.

6. Verify all requisite API are up on production, take help from infra team.

# Known Issues

N/A

# Reference Documents

|  |  |  |
| --- | --- | --- |
| Document Name | Description | Document Location |
| Babel Home | Scope and plan for Babel project. | <http://jira.moneycorp.local:8090/pages/viewpage.action?spaceKey=BAB&title=Babel+Home> |
| Environments set up | Hosting/Port information | <http://jira.moneycorp.local:8090/display/BAB/Azure+Environments> |
| Babel onboarding | Things you need to know in order to get started with Babel! | <http://jira.moneycorp.local:8090/display/BAB/Babel+Team+-+Onboarding> |
| API versioning | API versioning | <http://jira.moneycorp.local:8090/display/BAB/API+versioning> |