Service NSW

Omni Channel Reference Architecture

April 2019
Ver 1.2

Document History

Version	Author	Contact	Date	Changes
1.0	Michael Lisser	Michael.Lisser1@service.nsw.gov.au	July 18	Initial Unify Version
1.1	Michael Lisser	Michael.Lisser1@service.nsw.gov.au	March 19	Initial SNSW Version
1.2	Michael Lisser	Michael.Lisser1@service.nsw.gov.au	April 19	Refactored Version

Contents

1 Overview 6
1.1 Exec Summary 6
1.2 Business Objectives 6
2 Principles 8
2.1 Enterprise Principles 8
2.2 Application Principles 8
3 High Level Architecture 9
3.1.1 User Experience Components 10
3.1.2 Transaction Logic Components 10
3.1.3 Value Added Components 10
3.1.4 Services 11
3.1.5 Security 11
3.1.6 Auxiliary Capability 12
3.2 Context 12
4 Detailed Architecture 13
4.1 Overview 13
4.1.1 Component Deployment Model 14
4.1.2 Design Considerations 15
4.2 User Experience Components 16
4.3 Transaction Logic Components 16
4.4 Value Added Components 17
4.4.1 Transaction Logging 18
4.4.2 Metrics Logging 18
4.4.3 Receipting 19
4.4.4 SMS 19
4.4.5 Email 19
4.4.6 Notification 19
4.4.7 Encryption / Decryption 19
4.4.8 Case Management 20
4.4.9 Customer Management 20
4.4.10 Payment Management 20
4.4.11 JWT Management 20
4.4.12 Transaction Configuration 20
4.4.13 File Writer 21

4.4.14 File Reader 21
4.4.15 Caching 21
4.4.16 Error Mapping 21
4.4.17 Image Store 21
4.4.18 Signature Store 21
4.4.19 Pause / Resume 22
4.4.20 Proof of Identity 22
4.5 Services 22
4.5.1 Customer Services 23
4.5.2 Agency Services 23
4.5.3 Value Added Services 23
4.6 Security 23
4.6.1 Client API Security 24
4.6.2 Agency API Security 24
4.6.3 Data Security 24
4.6.4 Payment Fraud Detection 24
4.7 Auxiliary Capability 24
4.7.1 Environments and Organisations 24
4.7.2 Payments 25
4.7.3 Sensitive Information Handling 30
4.7.4 Agency Availability, performance and Caching 35
4.7.5 Large File Handling 35
4.7.6 Data Storage 36
4.7.7 Alerts 36
4.7.8 Digital Asset Management 36
4.7.9 CRM 37
4.7.10 Dashboards 38
5 Implementation Patterns 39
5.1 Single Channel Implementation 39
5.2 Multi Channel Implementation 39
5.3 Secure Multi Channel Implementation 40
5.4 Secure Multi Channel Implementation with Payments 40
5.5 Secure Multi Channel Implementation with Payments and CRM Integration 41
5.6 Full Environment 42
6 Future Direction 44

6.1 Fault Tolerance and Redundancy 44
6.2 Scaling 45
6.2.1 Scaling VAS Components 46
6.3 Databases 46
7 Example Implementations 47
7.1 RSA/RCG 47
7.2 IDP 47
7.3 CAR 48
7.4 Cost of Living 49

1 Overview

1.1 Exec Summary

This document presents a common Service NSW Omni Channel Reference Architecture as a design pattern for the delivery of new capability into the Service NSW ecosystem.

The aim of this document is to provide a pattern for delivery teams, products, projects and programs to work with so that they deliver a consistent solution across teams.

The aim of the architecture is to provide an Omni Channel approach to delivery that leverages allows autonomy within the teams, while ensuring conformity across programs of work, while leveraging new technology and a Microservice Architecture to deliver Customer Capability.

The architecture provides guidance and direction to Service NSW's internal Product and Project teams and our external partners (Agencies and Vendors).

It is expected that all product teams, and projects will conform to the approaches and guidelines laid out by this architecture.

It is anticipated that this architecture will evolve over time ensuring alignment between strategy and delivery.

1.2 Business Objectives

The Business Objective for the Omni Channel Reference Architecture is to achieve Omni Channel delivery of Services, and alignment of delivery.

Omni Channel delivery of services is where a service can be delivered equally across all channels, and a Customer can within a single transaction operate across multiple channels.

Omni Channel delivery does not imply that all transactions will be offered across all channels, some transactions will never be offered over the phone (simply due to the nature of the transaction), but rather that where a transaction is built in one channel it can be made available to all channels without major refactoring.

Multi Channel transaction operation is where a Customer starts a transaction in one channel, typically "saves" the transaction part way through, and continues the transaction in the next channel. For example a Customer may start a transaction on the Mobile, get part way through, save the transaction, and continue it on their desktop, get further through, save it again and complete the transaction at a Service Centre.

By Alignment of Delivery the business wish to see all programs, products and projects adopt the same pattern and structure of delivery, that reuse is encouraged, while autonomy of the team is maintained.

The Omni Channel Reference Architecture also provides a blueprint for the migration of capability of the legacy stack and onto the new technology platforms.

Overall the business desire a solution that is:

- Omni Channel
  - Able to deliver the same Business Capability across any channel equally
- Customer Centric
  - Able to deliver transactions with a focus on useability

- Technology Independence
  - Able to support a wide range of technology choices
- Consistent
  - Able to deliver the same look and feel across all channels
- Scalable, High Performant and Portable
  - Able to deploy and scale a high performance solution in a multitude of environments
- Low Cost for Delivery
  - Able to deliver the solution at a low cost without the need for specialist resources
- Low Cost for Licencing
  - Able to deliver the solution without the need for high cost specialist licences
- Secure and Auditable
  - Able to deliver a solution that secures data, access and capability in line with expectations of a government department
- Extensible
  - Able to be used equally across multiple transactions

2 Principles

The Architecture is developed in accordance with a number of overarching principles, these principles can be decomposed into two primary categories:

- Enterprise Principles
- Application Principles

Enterprise Principles guide the overall Service NSW Architecture.

Application Principles guide the delivery of a capability within the Service NSW Architecture.

2.1 Enterprise Principles

Enterprise Principles include:

Enterprise Patterns – The principle that all Products and Projects should follow a similar overarching enterprise design pattern.

Application Patterns – The principle that within the bounds of the Enterprise Patterns, that Product and Projects are able to design their own independent Application Patterns for implementation.

Change Patterns – The principle that changes to Architecture Patterns is allowed.

Collaboration – The principle that decisions on Change Patterns is made through collaboration.

Self Service – The principle of enabling self service delivery of platforms and components.

Reuse – The principle of reusing existing (new technology) capability in preference to rebuilding.

Omni Channel – The principle of delivering all transactions equally across all channels.

2.2 Application Principles

Application Principles include:

Future Proof – The principle of developing solutions that align with the SNSW Strategic Direction.

Encapsulation and Abstraction – The principle of grouping like functionality together, and accessing it via a simplified set of interfaces.

Singe Responsibility – The principle of automic transactions, where a transaction performs only a single purpose.

Continuous Delivery & Integration – The principle of deliver and integrate capability continuously.

Continuous Improvement – The principle of building on existing capability to continually improve solutions.

Don't Overengineer – The principle of keeping things simple.

Lightweight Pipes and Smart Endpoints – The principle that functional capability is only held in Component, not connections.

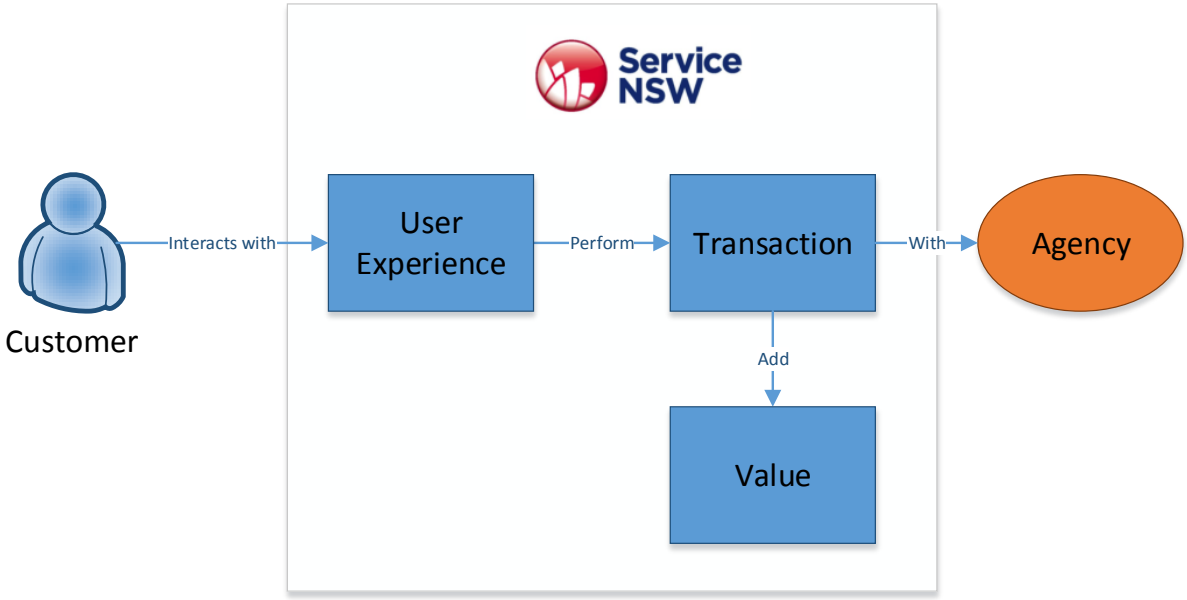
Microservices with Security – The principle of leveraging a microservice architecture overlayed with security at the API level.

Stateless – The principle of Stateless Transactions where state is only held at the agency.

Centralised Logging – The principle of leveraging a centralised logging solution for all components.

3 High Level Architecture

The high level architecture is based on a Microservice Strategy, where there is a clear separation between the channel delivery mechanism (User Experience), the underlying functional capability (Transaction Logic), and shared services (Value Added).



User Experience – The channel technology involved in delivering the experience (look and feel) to the Customer.

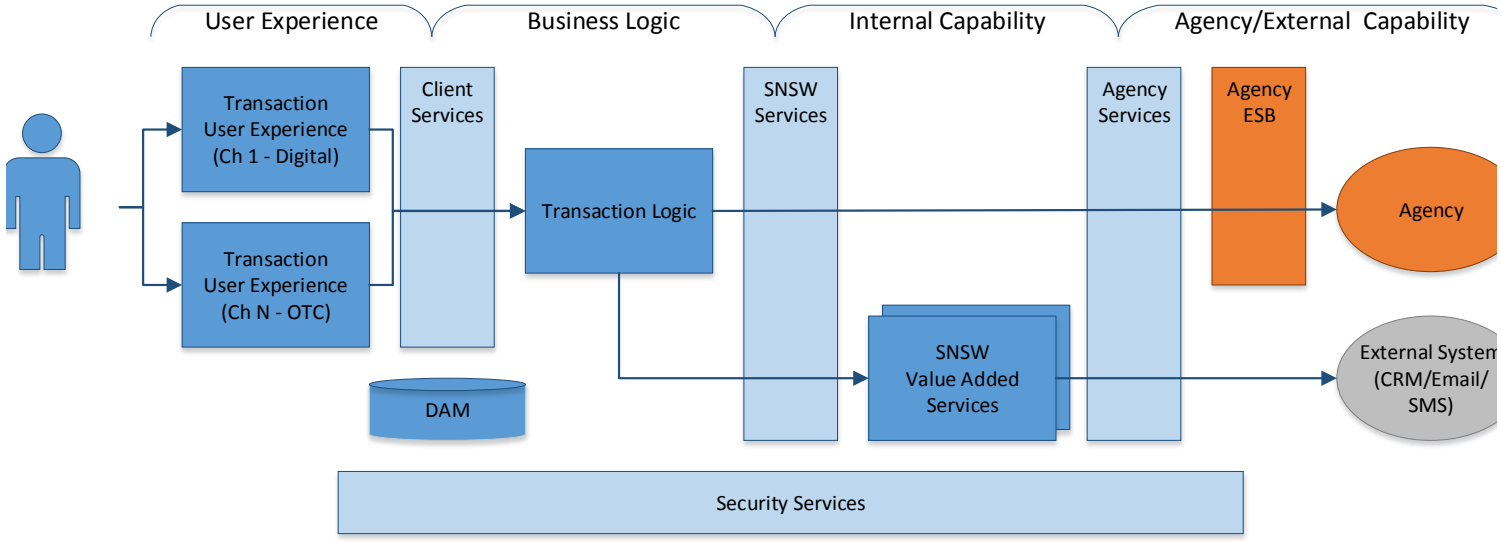
Transaction Logic – The process involved in executing the transaction with the Agency.

Value Added – The services that SNSW brings to any transaction (such as Email, SMS, Receipting, Payments etc).

These core building blocks are abstracted through the use of a Service Layer, and underpinned through the use of a common security platform.

To gain a level of conformity across the Channel Experiences a Digital Asset Management platform is used to house and deliver Digital Assets.

Each of the components are intended to be stateless.



3.1.1 User Experience Components

The User Experience is the primary point of contact between the Customer and the Service NSW offerings. The User Experience resides in the DMZ, outside the Service NSW firewall (in front of the Client Services).

The User Experience is intended to provide a light weight presentation layer to any transaction.

The User Experience is intended to be consistent across transactions (similar look and feel, function etc). This consistency of interface is achieved through the leverage of a Digital Asset Management platform.

It is intended that for any given Transaction there are multiple User Experience Components.

The User Experience is abstracted from the Transaction Logic through Client Services.

3.1.2 Transaction Logic Components

The Transaction Logic is the primary implementation of the business logic to execute a transaction with an agency (or agencies). It is a headless component accessed only via Web Services. The Transaction Logic orchestrates the process of completing the transaction with the agency, along with any additional capability (Value Added) layered on top of the transaction by Service NSW (such as Taking a Payment, Sending an Email, Generating a Receipt etc).

The intent is that there is only one Transaction Logic component per transaction. There are expected to be 1,200 transactions provisioned through SNSW, which equates to 1,200 Transaction Logic Components. It is worth noting the Transaction Logic component may offer multiple services to the User Experience to complete a transaction (such as Price Transaction, Validate Data, Complete Transaction, Roll Back etc).

The Transaction Logic is intended to be stateless.

The Services offered to the User Experience should be clearly defined business services.

3.1.3 Value Added Components

Value Added Components provide discrete common capability that can be leveraged and re-used by any transaction logic. Value Added Components are headless and accessed through their APIs.

Value Added Components can only be accessed via Transaction Logic as part of a business process, they are not intended to be used directly by a User Experience Component.

The Value Added Component is independent of any specific Transaction.

Typical Value Added Components include:

- Receipt Generation
- Email Messaging
- SMS Messaging
- Transaction Logging
- Payments
- Case Management
- Client Management
- File Generation/Transmission
- Pause/Resume

It is anticipated that there will be about 20-30 Value Added Services.

3.1.4 Services

The Service Layer provides abstraction and security between the core components.

The Service Layer is intended to be a light weight dumb pipe with smart end points. That is it implements no actual capability in relation to any transaction logic or value added capability.

3.1.4.1 Client Services

Client Services expose Transaction APIs to the end User Experience Layer.

The Client Services need to implement and enforce any B2C security required on the APIs exposed.

The Client Services only expose Transaction APIs.

3.1.4.2 Agency Services

Agency Services expose and standardise services provisioned by the downstream Agency.

Agency Services provision the B2B security applied between SNSW and the Agency.

Agency Services may re-implement Agency Services into a more SNSW friendly standard to ensure consistency across multiple agency APIs.

3.1.4.3 Value Added Services

Value Added Services expose the Value Added components to the Transaction Logic.

The Value Added Services do not need to apply security as the APIs are not exposed outside the SNSW network.

3.1.5 Security

Security is leveraging the existing SNSW security patterns, particularly focusing on:

- Identity Management
- Client Edge Security B2C
- Business Security B2B

Identity Management is provisioned by the SNSW Identity Management Platform, and allows the enforcement of Credentials and Authentication. This is provisioned through an OAuth Standard.

Client Edge Security needs to be applied at the Client Services Layer, where access to an API is only available if the user has first authenticated. This is a user based security approach.

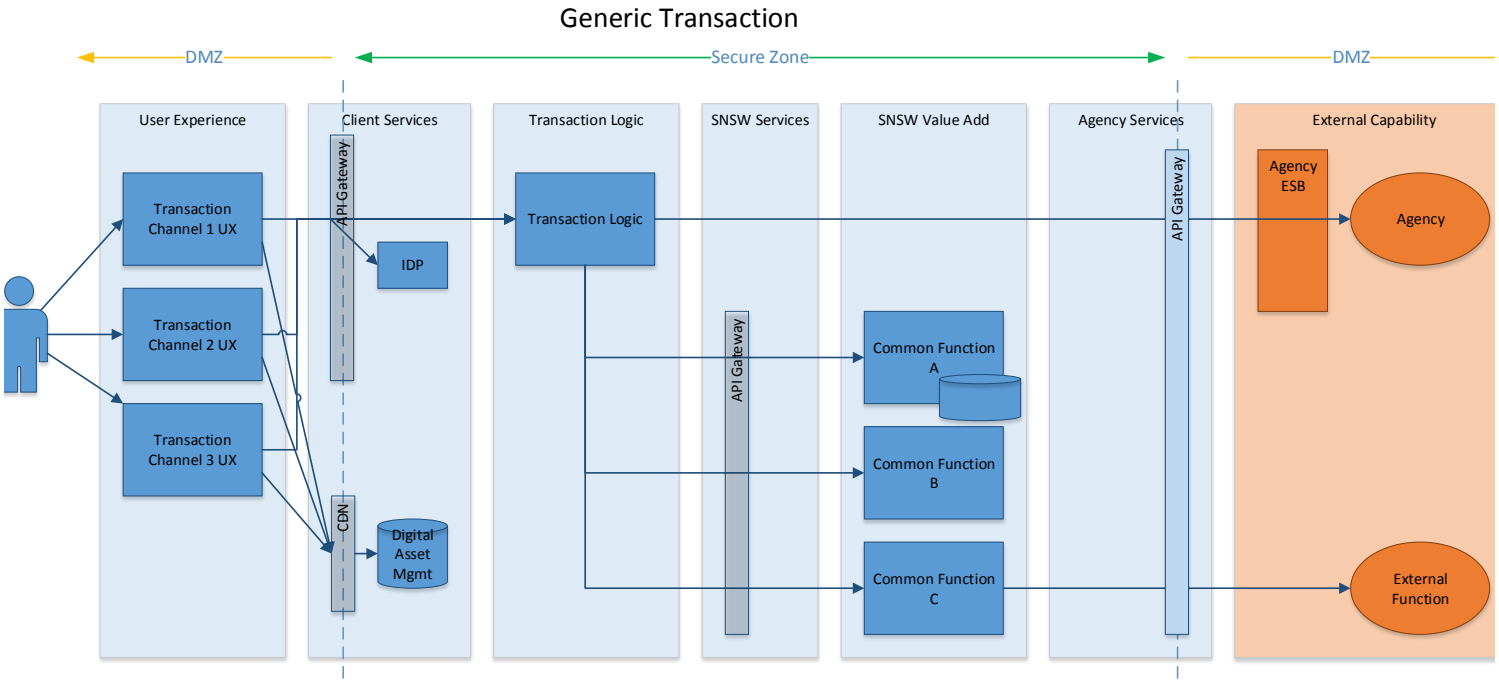
Business Security is applied between SNSW and the downstream agency. Typically this is a variation of TLS, where certificates are used and exchanged between SNSW and the Agency rather than user specific authentication/authorisation.

3.1.6 Auxiliary Capability

Supporting the overall architecture are a number of supporting components specifically worth noting are:

- **Digital Asset Management Platform** – for the management and provision of Digital Content to the Client UX Components.
- **Cloudfron CDN** – for the caching an high speed provision of digital content contained within the DAM.
- **AWS Database** – for the storage of long term data across components and transactions.
- **API Gateway** – for the provision of the service layer abstractions

3.2 Context



4 Detailed Architecture

4.1 Overview

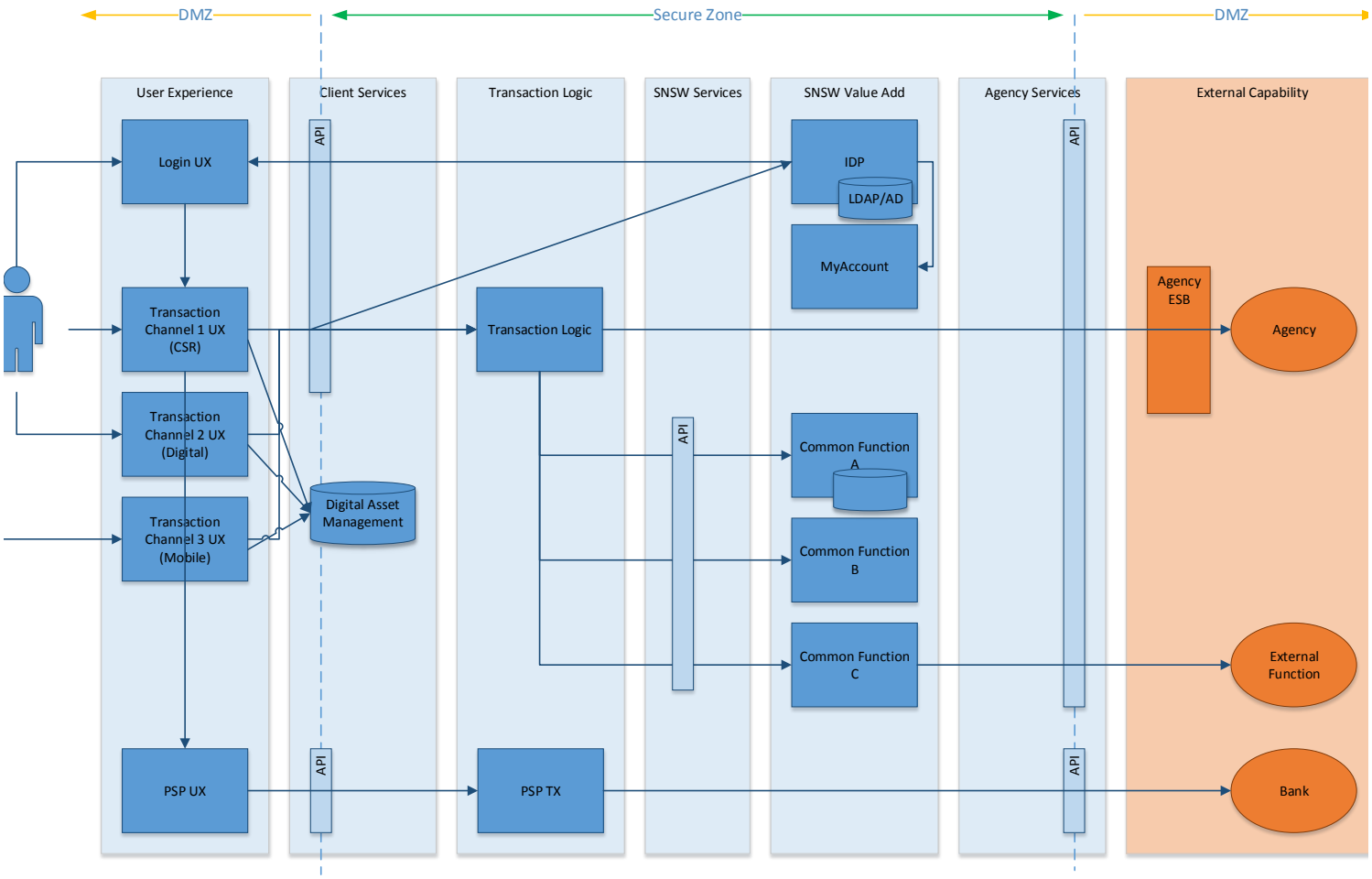
The Omni Channel Reference Architecture decomposes the solution into seven distinct layers.

- User Experience
- Client Services
- Transaction Logic
- SNSW Services
- Value Add
- Agency Services
- External Capability

With a given transaction supporting multiple distinct Client Channels, and leveraging existing technology platforms such as:

- PSP – Payments
- IDP / MyAccount – Identity Management

Secure Multi Channel Transaction with DAM and Payment



4.1.1 Component Deployment Model

The physical deployment is different to the conceptual model presented in the previous section.

The new architecture solution is based on a componentised microservice architecture where components are built and deployed into PCF which in turn resides in AWS.

Within AWS is the PCF environment. This environment is split into two sections:

- External PCF – Components that are public facing (outside the SNSW Firewalls)
- Internal PCF – Components that are internally facing (inside the SNSW Firewalls)

The main difference between an External and Internal PCF component is the Routing Path through the PCF Load Balancers. Any component that is Externally facing has an External Routing Path (ie https://unify.pcf-ext.service.nsw.gov.au). Any component that is not explicitly External Facing is internal facing and has an internal Path (ie https://unify.pcf.service.nsw.gov.au).

Internally Facing PCF components can only be accessed via the API Layer. The API Layer exposes the internal components to the external Clients via a secure API gateway (where required). Thus all interaction between an External PCF component and an Internal PCF component is via the API Layer.

The API Layer applies and enforces security and routing. It enforces which components can be accessed by which clients.

Within the AWS environment are some specialist services that specifically reside outside the PCF environment (by design). Primarily:

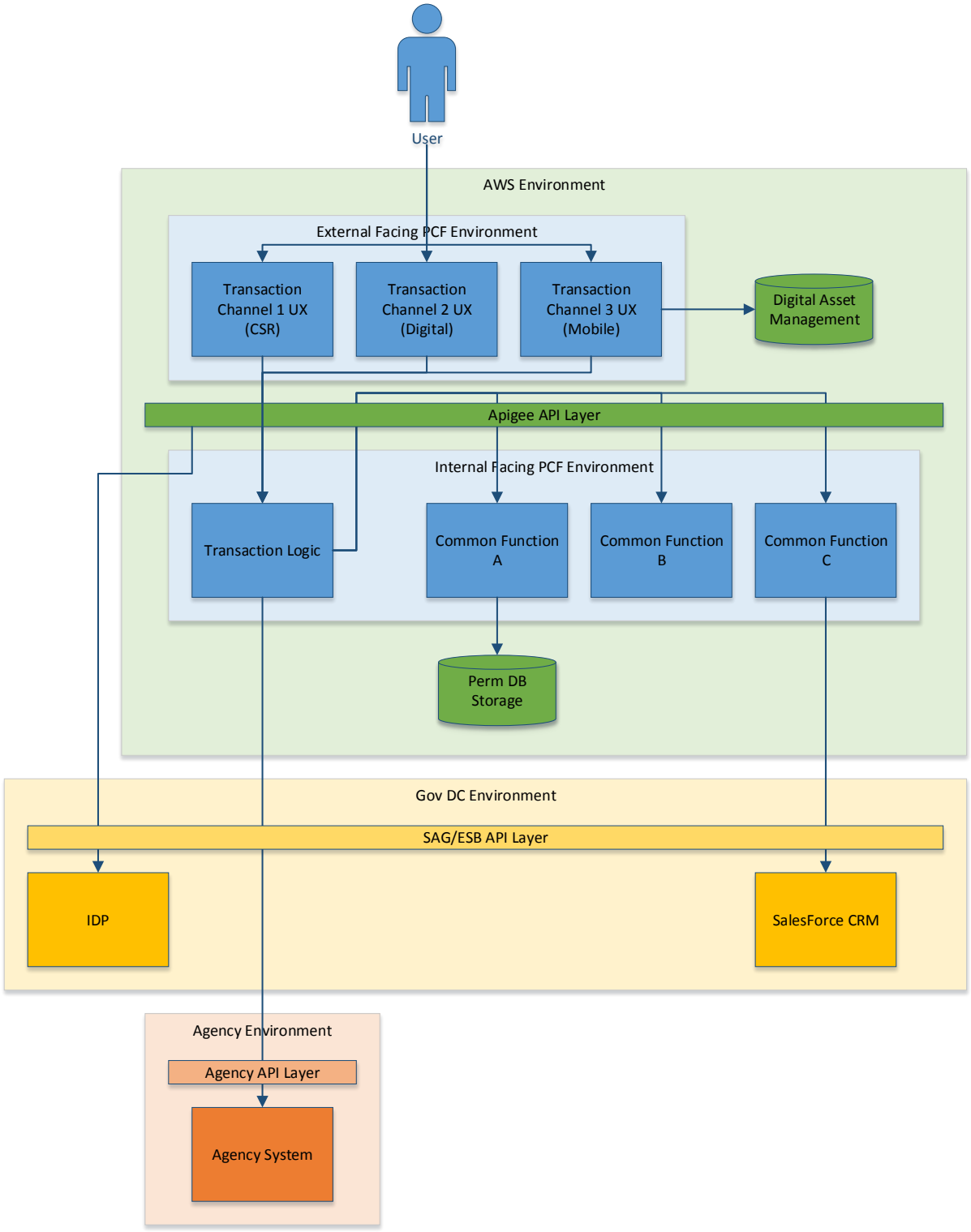
- Permanent DB – For storage of secure records
- Digital Asset Management – For the provision of static data.

The DAM resides behind a high speed AWS Data Cache (CDN) known as Cloud Front.

There is still integration to the existing SNSW Infrastructure via Gov DC. This allows the PCF/AWS platforms to leverage the existing and legacy implementations and services (such as Identity Management, and Integration to existing agency and systems such as CRM).

With the integration to downstream agencies, there are two approaches. Integration through an existing API exposed through the legacy SAG/ESB environment (depicted below), or integration directly to an Agency API exposed through the Apigee API Layer (not depicted below).

Thus the physical implementation can be depicted:



4.1.2 Design Considerations

When developing to this architecture the following design considerations should be observed:

- **Client Resides in the DMZ** – Thus no critical or sensitive data should be held in the Client.
- **Use of the DAM** – Any data likely to change over time should be held in the DAM.
- **Security Applied at the API Layer** – All access control to APIs should be handled in the API layer.
- **Light Weight Pipes** – No functional capability should be developed in the API layer.

- **Transaction Identification** – All interactions across a transaction should utilise the same Transaction Id.
- **Logging** – All inputs, outputs, and errors for any Web Service should be logged.
- **Logging Source** – All client machine identifying information should be logged.
- **Logging Metrics** – All Web Service Transaction Times should be logged.
- **Operational Data** – All operational data should be held in a transaction operational data store.
- **Scalability** – All components can be independently scaled to improve performance.

4.2 User Experience Components

The User Experience Component provides a physical User Experience on a specific transaction. The capability of the User Experience Component will greatly depend on the capability provided by the underlying transaction component and the needs of the Customer.

The User Experience Component is intended to be light weight (throw away) with little scope for complex logic or business processing (all business processing is to occur in the Transaction Logic).

The intent is that the UX is fast to build, quick to deploy, independent of anything else, and so easy to replace. The UX Component is technology agnostic, in that one UX Component has no relationship to the next.

Where UX specific capability is required, this should be developed into the UX Component and not the Transaction Logic.

To provide consistency across UX Components a DAM is provided and able to be leveraged. The DAM provides JSON APIs to allow access to content stored in the DAM. The amount a UX Component leverages the DAM is up to the developers of the component. Typically content that is expected to change over time, and does not impact the delivery of the capability should be sourced from the DAM, such as Help Text, Welcome Messages, Terms & Conditions, even Style Sheets.

Where the UX Component requires access to secure capabilities or services the UX Component must Authenticate the User against SNSW's IDP.

The UX Component resides outside the Secure Zone in SNSW, in the DMZ, as such it must be considered to be "compromised" at all times from a security perspective. The UX Component therefore must not contain any secure keys, passwords, or processing.

The UX Component will typically integrate to the Transaction Logic via the API Layer utilising only the Client Services exposed by the API Layer. This integration is via REST Web Services.

4.3 Transaction Logic Components

The Transaction Logic Component utilises a number of Agency Services, and Value Added Services to complete a transaction for the calling client.

The Transaction Logic Component provides a centralised processing capability for the transaction being undertaken. Typically each transaction will operate in its own independent Transaction Logic Component. Typically each Transaction Logic Component will provide a number of REST Web Services to allow execution of the transaction.

The Transaction Logic Component is intended to be channel agnostic. No web service should be offered that requires knowledge of the calling client or channel. Typically web services will follow a similar process.

- Get Setup Data
- Initiate a Transaction
- Validate Data
- Take a Payment (via the PSP)
- Complete Transaction with Agency
- Generate a Receipt

While the exact steps offered for any specific transaction will vary, the overall flow is expected to be similar across transactions.

There should be some underlying consistent behaviour across all Transaction Logic Components.

- Generation and Use of a Transaction Id
- Logging of Inputs and Outputs
- Logging of Metrics

A Transaction Id should be generated at the start of processing any transaction, and passed back and forth across all subsequent calls relating to that transaction to allow for tracking and logging.

The Transaction Logic Component is intended to be light weight and stateless. There should be little to no data storage required. Where data is persisted it should be in one of two forms:

- Long Term Audit Data – Stored into a permanent external data store
- Short Terms Operational Data – Stored into a temporary data store

The Long Term Audit Data is write only data stored into a centralised DB in AWS hosted outside of the PCF environment. This data is intended to be held past the life of the transaction component.

The Short Term Operation Data is read/write data stored into a local DB generated within PCF (though hosted in AWS). This data is only relevant to the transaction being executed or for a short time afterwards.

4.4 Value Added Components

The Value Added Components provide shared cross transaction services that can (if needed) be used by any Transaction Logic Component.

The Value Added Component is a headless web service accessed only via a limited set of Rest Service APIs.

Internally the component is intended to be a stateless system that provides a single targeted capability.

There is only expected to be a relatively small number of Value Added Services (things Service NSW adds to a given transaction).

Currently the following Value Added Services are either in place or underway:

Existing VAS Components

- Transaction Logging

- Metric Logging
- Receipting
- SMS Messaging
- Email Messaging
- Encryption/Decryption
- Case Management
- Customer Management
- Payment Management
- JWT Management
- Transaction Configuration
- File Writer

Planned VAS Components

- Caching
- Error Mapping
- File Reader
- Image Store
- Signature Store
- Pause/Resume
- Proof of Identity

4.4.1 Transaction Logging

The log is “Write Only”, where data can only be added to the log, never updated, modified or deleted.

In the initial deployment the logs are write only, with no capability for read. This protects any data held within the log as there are no enabled “read” functions to access the data.

In future releases data held within the log file will need to be encrypted. Particularly if transactions are logging sensitive or personal information. In this situation the core metadata may remain unencrypted, allowing access, sorting, and analysis, while the data can be encrypted, thus protecting it against unauthorised access.

In the Transaction Log where the log is required to store a document as part of the log, the document is split from the log and stored in a separate table. Hence one call to the Tx Log may result in two different table entries.

The mechanism used for storage of the log data itself is a MS SQL Server DB held in AWS.

4.4.2 Metrics Logging

Metrics Log – is only associated with the timing and execution of transactions, and should have recorded in it the start and end time of a transaction through any key component of the system.

The log is “Write Only”, where data can only be added to the log, never updated, modified or deleted.

In the initial deployment the logs are write only, with no capability for read. This protects any data held within the log as there are no enabled “read” functions to access the data.

In future releases data held within the log file will need to be encrypted. Particularly if transactions are logging sensitive or personal information. In this situation the core metadata may remain unencrypted, allowing access, sorting, and analysis, while the data can be encrypted, thus protecting it against unauthorised access.

The mechanism used for storage of the log data itself is a MS SQL Server DB held in AWS.

4.4.3 Receipting

The receipt generator takes input from the calling transaction and converts this input into a Service NSW PDF based receipt (using SNSW letterhead and copy etc).

The mechanism used for generation of the PDF is currently under investigation and still to be determined.

The result of the process is that a PDF file is stored in local storage against the Customer and Transaction, and returned to the calling transaction.

Where a copy of a receipt is requested, a previously stored receipt is retrieved and returned to the calling transaction.

The mechanism used for storage of the receipt data itself is through a MS SQL Server DB held in AWS.

As the receipt may record sensitive and personal information the data held in the receipt data store may require encryption.

4.4.4 SMS

The SMS Notification component allows the calling transaction to send SMSs to the Customer.

The mechanism is based on sending the SMS to a known Mobile Number.

SMSs are sent via the existing Twillio Gateway.

4.4.5 Email

The Email Notification component allows the calling transaction to send emails to the Customer.

The mechanism is based on sending the email to a known email address.

Emails are sent via the existing SendGrid Gateway.

4.4.6 Notification

This component allows the sending of Messages to the Customer based on the Customers existing profile.

The component should be passed the Message, Transaction Type, and the Customer Id, where it should lookup the Customer within Service NSW, determine the Customers Messaging Preference (Email, SMS, Both, None) for the given Transaction Type, and send the message using that preference.

4.4.7 Encryption / Decryption

The encryption capability allows the encryption and decryption of data stored in secure data stores. The services offered are:

- Encrypt Data
- Decrypt Data

The algorithm used for Encryption/Decryption will be specified by the security team.

The component does not store the encrypted data, it is up to the calling transaction to store the encrypted/decrypted result if needed.

4.4.8 Case Management

The Case Component allows the maintenance (search, read, creation, update, delete etc) of Case Data.

Case Data is the domain of SalesForce, however management and display of this information can occur within any transaction, or dashboard.

4.4.9 Customer Management

The Customer Component allows the maintenance (search, read, creation, update, and delete/terminate) of customer data.

There are many sources of Customer Data, which can generally be partially sourced from SNSW directly, or from each of the target agencies. This component allows the retrieval (and management) of Customer Data both known to SNSW and a specific agency in a uniform way.

4.4.10 Payment Management

The Payments Component allows the collection of CC and Cash payments over the counter (or digitally) through the OneGov GLS Payment System.

The Component is required to support Payments, Refunds, and Reconciliation.

On completion of a payment the component should redirect the user to complete the transaction.

It is envisaged that the payment component will utilise the existing PinPads (and Cash Draws) on the CSR Work Station, and not require the manual entry of CC Details.

4.4.11 JWT Management

The generation of JWT Security Tokens should be handled centrally to ensure that all transactions utilise the same approach for Security Management.

The component should allow the generation and retrieval of JWTs.

4.4.12 Transaction Configuration

The Transaction Configuration component allows the retrieval of Transaction Setup data (such as Drop Down List values).

This information can be sourced from different locations depending on the transactions requirements.

For Service NSW related data, this information is sourced from a temporary data store.

For Agency related data, this information is sourced from the downstream agency.

On calling this capability the caller must supply the transaction type to allow the identification of the correct setup data.

4.4.13 File Writer

Allows the writing of a file (Text, CSV, XML, JSON, Doc, PDF etc) to a defined location (S3 bucket).

4.4.14 File Reader

Allows the reading of a file (Text, CSV, JSON, XML, PDF, Doc etc), from a defined file location (such as an S3 bucket).

4.4.15 Caching

Caching allows a transaction to take advantage of a temporary storage space (a cache).

- Data can be pushed into the Cache.
- Data can be retrieved from the Cache.
- Data is segregated within the Cache based on Transaction Type.

Note: the cache will be provisioned through temporary storage, and so can go unavailable without notice.

Note: the cache is not intended to support wider "Search" or "Update" capability, simply write/retrieve based on a Transaction Type and Id.

4.4.16 Error Mapping

The Error Mapping component maps a provided (Agency) error message to a Customer Centric Error Message. It relies on a defined set of error message maps which is stored in a temporary database.

When the component receives a request the calling Transaction Logic component should provide the Error Message, Error Id, and Source System/Transaction. If an existing Map for this error message is found in the Mapping database then the associated error message is returned. If no map is found in the Mapping database then a default message is returned (the equivalent of “Oops something went wrong please try again later”, and a new entry is added to the database for later remediation).

4.4.17 Image Store

The image store contains encrypted copies of the Customer Photo.

The intent is that this image is reusable across all customer interactions and transactions where a photo is required.

4.4.18 Signature Store

The signature store contains encrypted copies of the Customer Signature.

The intent is that this signature is reusable across all customer interactions and transactions where a signature is required.

4.4.19 Pause / Resume

Provides the capability to Pause and Resume a transaction mid flight.

This capability supports the services:

- Pause Transaction
- Resume Transaction

It is intended to be used for transactions where there is a high likelihood of a “walkout” such as during a manual test (where there is a natural break in a transaction), before a payment is made, where the customer can leave and return at a later date.

The component should allow the transaction state to be written to temporary storage and retrieved at a later date. On retrieval the transaction data should be removed from the temporary storage. Equally the temporary storage should only house data for a maximum set period (suggest 5 days).

In writing the transaction to the temporary storage some core metadata is required outside that of the transaction data to allow correct identification at a later date. This includes:

- Transaction Id
- Transaction Type
- Transaction Date
- Transaction Name
- Customer Id

4.4.20 Proof of Identity

POI is the conglomeration of a proof of identity function, where a Customers Image, Signature, and Identity credentials can be stored by SNSW and used at a later date in other applications. Thus allowing applications requiring POI to be performed online, and streamlining the process for future CSR transactions.

Note this can not be done without the customers consent.

The function relies on creating secure data stores, so relies on Encryption, and storage of Image, Signature and POI information.

The POI component should store the identity level that the Customer has been certified to, these being:

- Uncertified
- Document Certified
- Visually Certified

And the storage of the Point System associated with the certification.

This data must be encrypted prior to storage, and decrypted on retrieval.

4.5 Services

The Microservice Architecture relies on utilising Web Services.

In this architecture approach the Web Services are:

1. Light Weight
2. Stateless
3. Encapsulated

That is the web service interfaces are simple, independent, and self contained. In this environment they are all REST Web Services.

There are three primary groups of Web Services:

- Customer Services – Services exposed by SNSW to an external Customer UX component.
- Agency Services – Services exposed by a downstream Agency to SNSW
- Value Added Services – Internal SNSW capability used by Transaction Logic Components.

All the web services are intended to be Stateless.

All the web services are intended to be REST Web Services.

4.5.1 Customer Services

Customer Services provide the entry point for a specific channel (or external client) to call a Transaction provided by Service NSW.

The Customer Services is required to implement B2C authentication on all sensitive Web Services, where the Web Service must check the Customer has Authenticated.

The Customer Service must also check that the data being accessed is appropriate to the Customer who has Authenticated. That is Customer A has authenticated as A but attempts to access Customer B's data (this should be prevented).

The exact services exposed to the Client via a Customer Service will greatly depend on the underlying Transaction being exposed.

4.5.2 Agency Services

Agency Services provide access by internal SNSW components (specifically Transaction Logic components) to access agency capability.

The Agency Service is required to implement B2B security working in conjunction with the downstream agency to implement a TLS style of security.

The Agency Service is also require (where necessary) to translate the External Agency APIs to an internal REST API.

4.5.3 Value Added Services

Value Added Services provide access to internal Value Added Components.

These services are completely internal to SNSW and so able to be accessed from Transaction Logic via the API layer directly. As these components are not exposed to external systems there is no security requirements on accessing the APIs.

4.6 Security

Security is applied at multiple levels.

4.6.1 Client API Security

Client API Security requires the use of OAuth credentials. Where a Client Calling Component is required to Authenticate the User prior to making calls against the API layer to access Client Services.

Authentication of the client will vary depending on if the Client is an External Customer, or an internal Staff Member.

- External Customers are authenticated against their credentials in LDAP.
- Internal Staff are authenticated against their credentials in AD.

Once the client is authenticated, a secondary check is required at the API layer to ensure the Client is only accessing their own data. This is critical for Public Customers, where it may be possible to Login as Client A, but then try and access data for Client B (by say fudging data calls). The API layer must protect against this kind of data tampering.

4.6.2 Agency API Security

Agency API Security requires the use of mutual exchanges certificates. Where data is transferred over a secure channel to the agency using SSL, and TLS. In this scenario data is transferred in a B2B transfer protocol and so is Customer Agnostic.

4.6.3 Data Security

In Flight Data Security requires the use of encryption for all inflight public data channels using SSL.

At Rest Data Security only occurs in a limited number of scenarios. The PCF Architecture does not support File System (file writing) so at rest data retention is limited.

The only significant data store in use is in relation to Audit and Metric Logging (Value Added Services). These components provide Write Only access to an external Database in AWS. The External Database is contained within the SNSW environment and has no “Update” capability. Long term the payload elements of this data need to be encrypted where personal information is involved. At present this is not occurring due to the need to manage a bedding down process for new application delivery. Once established Encryption of Personal Information should be enforced.

Some programs are continuing to write data to SalesForce in the form of Case Data. This information is governed by the SalesForce data management policies.

4.6.4 Payment Fraud Detection

When executing transactions that require payments, the system verifies that the Transaction Id, and Payment Id passed into the Transaction Logic Component match that held by the Payment System, and that the valid correct amount has been collected prior to execution of the Transaction.

4.7 Auxiliary Capability

The following Auxiliary Capability extends the basic Microservice Architecture to extend and enhance the capabilities of the overall platform.

4.7.1 Environments and Organisations

The Pivotal Cloud Foundry environment provisions two physical environments and multiple virtual environments:

- Production
    - External Rout

    - Internal Rout
- Development
    - Dev
        - External Rout
        - Internal Rout
    - Test
        - External Rout
        - Internal Rout
    - Integration Test
        - External Rout
        - Internal Rout
    - Load
        - External Rout
        - Internal Rout
    - PSM
        - External Rout
        - Internal Rout

Any component provisioned with an External Rout will be accessible from the Public Internet, any component provisioned with an Internal Rout will only be accessible from within the SNSW network.

Within these environments the components are grouped by Organisation. To date the Organisations have been arranged by Platform and Project. With the intent being to share capability the organisations now need to be arranged around the architecture:

- Channel Experience
- Transaction Logic
- Value Added Services

Channel Experience represents the components that are used to delivery the Customer Experience. These are typically specific to a unique channel.

Transaction Logic represents the shared transactions that are accessible across all channels.

Value Added Services represents the set of Value Added Components that are shared amongst Transactions.

4.7.2 Payments

With the introduction of a Microservice Architecture and the move away from a monolithic transaction solution driven by SalesForce, there needs to be an examination of the method of processing payments.

The current SalesForce solution for processing payments, relies on SalesForce storing and holding state, while the payment is processed via the PSP system, and then re-executing the transaction based on the saved state. This is in essence a pause/resume process in the transaction processing flow.

With the introduction of a Microservice Architecture there is no state held by the transaction processing system. The transaction processing system is no longer a monolithic application, rather a multi threaded microservice with no perceived state.

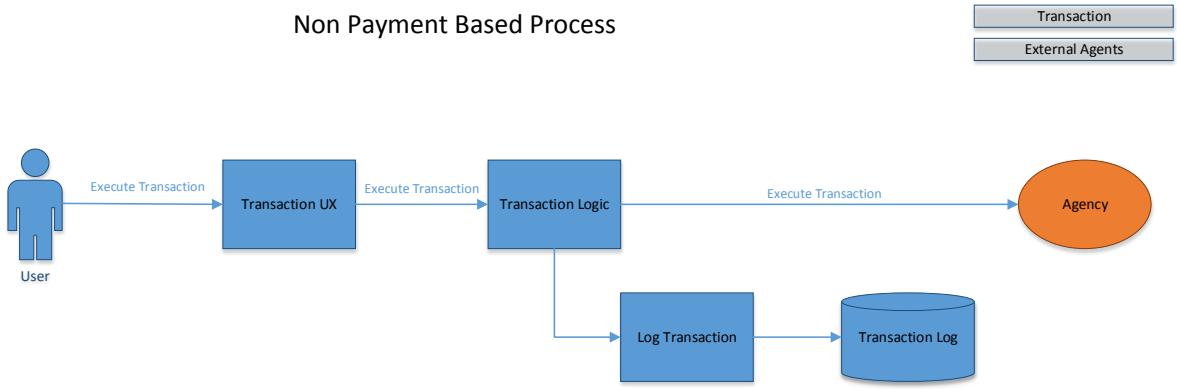
The current PSP implementation relies on state being held by the transaction processing system to ensure fulfilment and reconciliation. Without state being held by the transaction processing system there is a high likelihood of discrepancies and failures in the processing of payment transactions, causing increased load on reconciliation processes.

Storing state in a large distributed microservice architecture is undesired, as it impacts all transactions reduces reliability and stability of the overall network.

4.7.2.1 Solution Overview

In a general non payment transaction, the user (Customer or CSR) interacts with a front end screen (User Experience) to preform a transaction. The User Experience communicates with back end Transaction Logic to perform the transaction with a downstream Agency. In the process of executing a transaction, the transaction is logged to a system of record. There is no state held by any component within SNSW, and so the solution is highly scalable and performant.

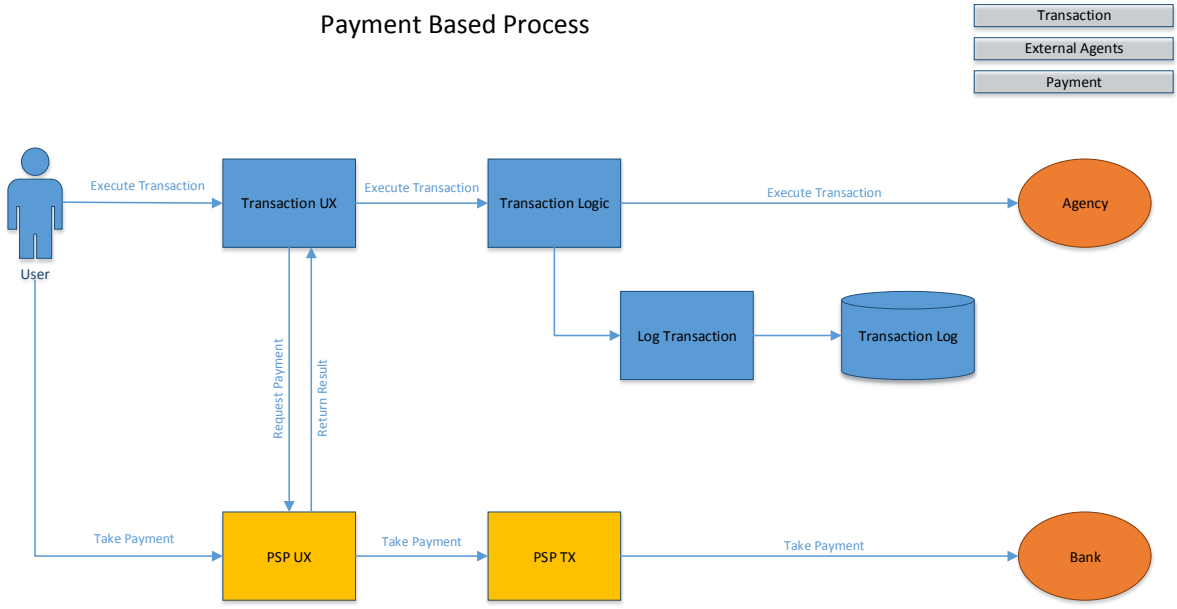
Non Payment Based Process



When moving to a transaction involving a Payment, the overall process should remain consistent, stateless.

The current proposed solution is for the Transaction UX to redirect to the Payment UX, allow the User to provide their payment details, process the payment via the PSP and then return to the Transaction UX.

Payment Based Process



On the whole this process seems sound. However when switching from the Transaction UX to the PSP UX there is a real chance that the return call is never completed from the PSP UX (say the Customer closes their browser), thus leaving the system in a state where a Payment is Taken for a Transaction that is never executed.

The Payment team have realised this deficiency and also returned a separate “Notification” of payment (not shown) however if control is never returned to the Transaction UX (say the Customer closes their browser) there is nothing for the Notification service to call that knows of the transaction.

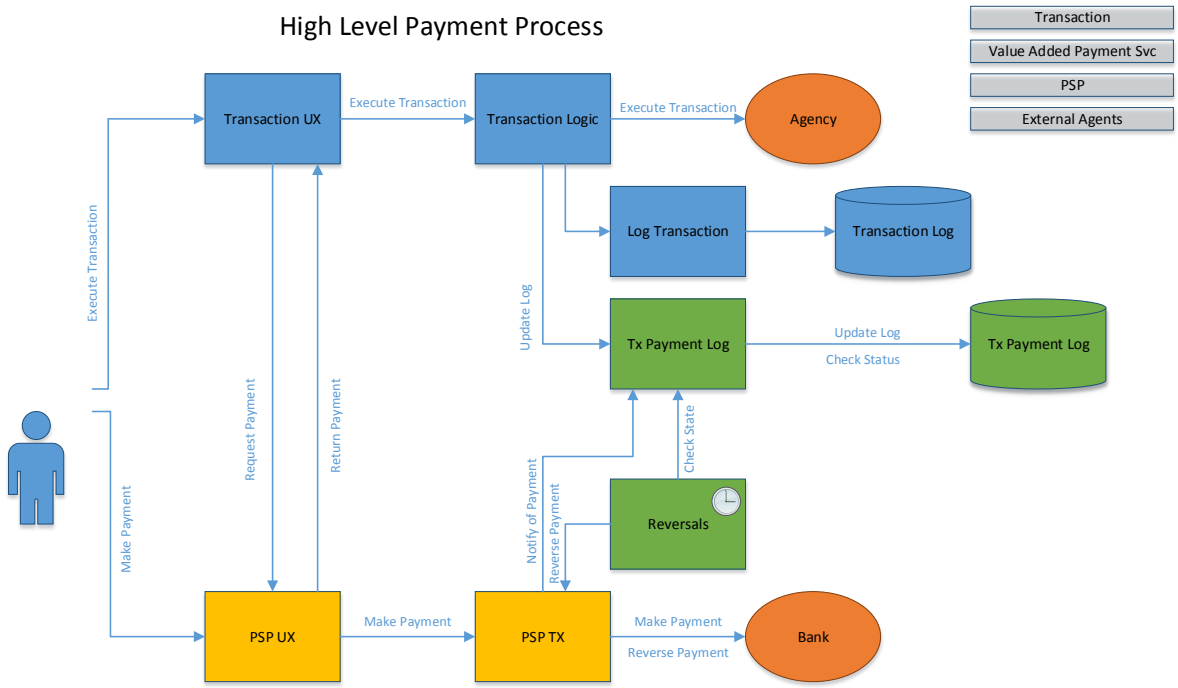
It was proposed that a two phase commit in the Payment System would resolve this issue, however time does not permit such a solution, and there are outstanding technical reservations as to how best to implement.

It was also proposed that storing state in the transaction would resolve this issue, however this invalidates the stateless microservice architecture approach now being adopted, and would place significant overhead on each transaction being delivered (as there are over 1200 transactions to be delivered).

4.7.2.2 Implemented Solution

The implemented solution makes use of the Payment Solution as it exists today, but leaves the transaction system (in effect) stateless.

The proposal is to introduce a new ValueAdded Service for Payment Processing into the Microservice Architecture. This service would maintain state of a transaction payment outside the flow of the transaction, while a second periodic process would then use this information to reverse any erroneous transactions inter day.



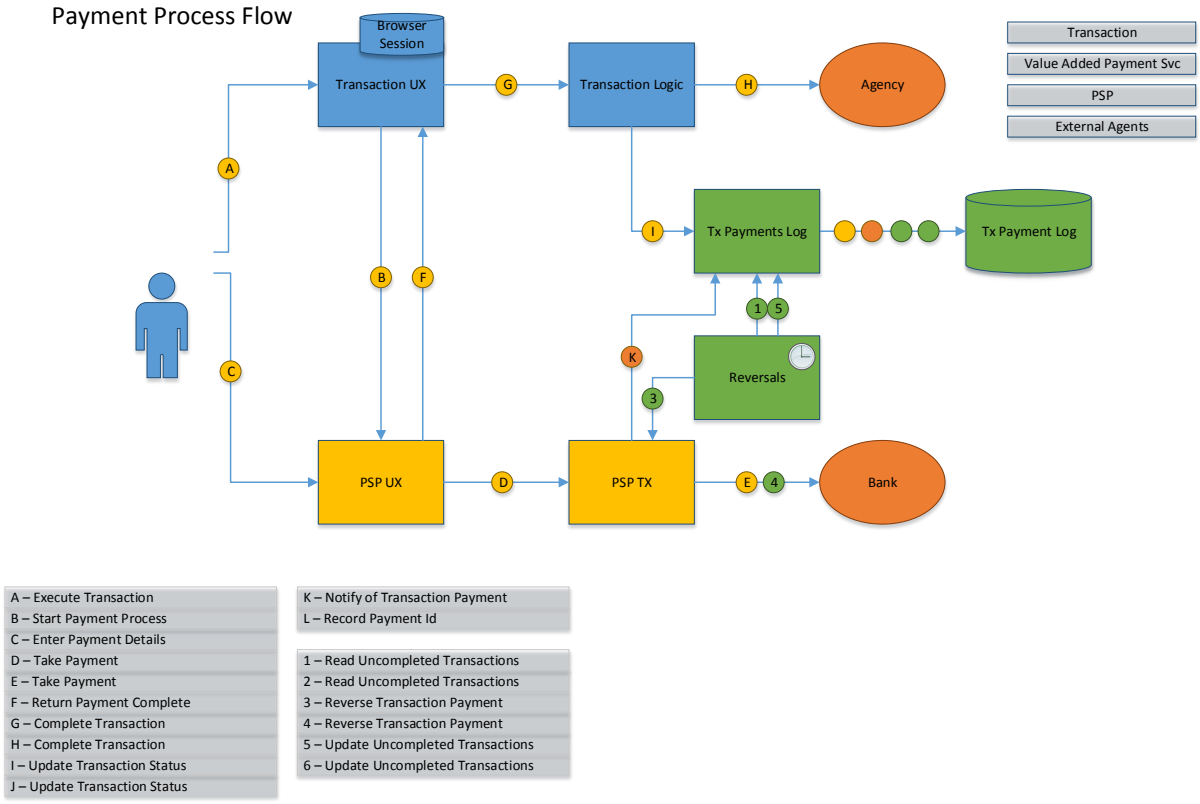
From a transaction processing perspective the new Payment Log would be a write only log, where each new transaction involving a payment would be written to, logging only core information (such

as Tx Id, Payment Id, and Tx Status). As this is a write only pattern no state is held by the Transaction logic.

When the PSP accepts a payment, a Notification is sent to the calling system. In this situation all notifications would be routed to the same Payment Log. This message would also include the Tx Id, and Payment Id, however would not contain the Tx Status.

Thus over time the Tx Payment Log would contain a list of all completed transactions, along with their payments, and equally a list of all transactions that were paid for but never executed (those without a valid status).

Periodically a Reversal process would run, searching the Tx Payment Log for incomplete transactions (where a payment was made but no transaction was completed) and using the existing PSP services revers the payment.



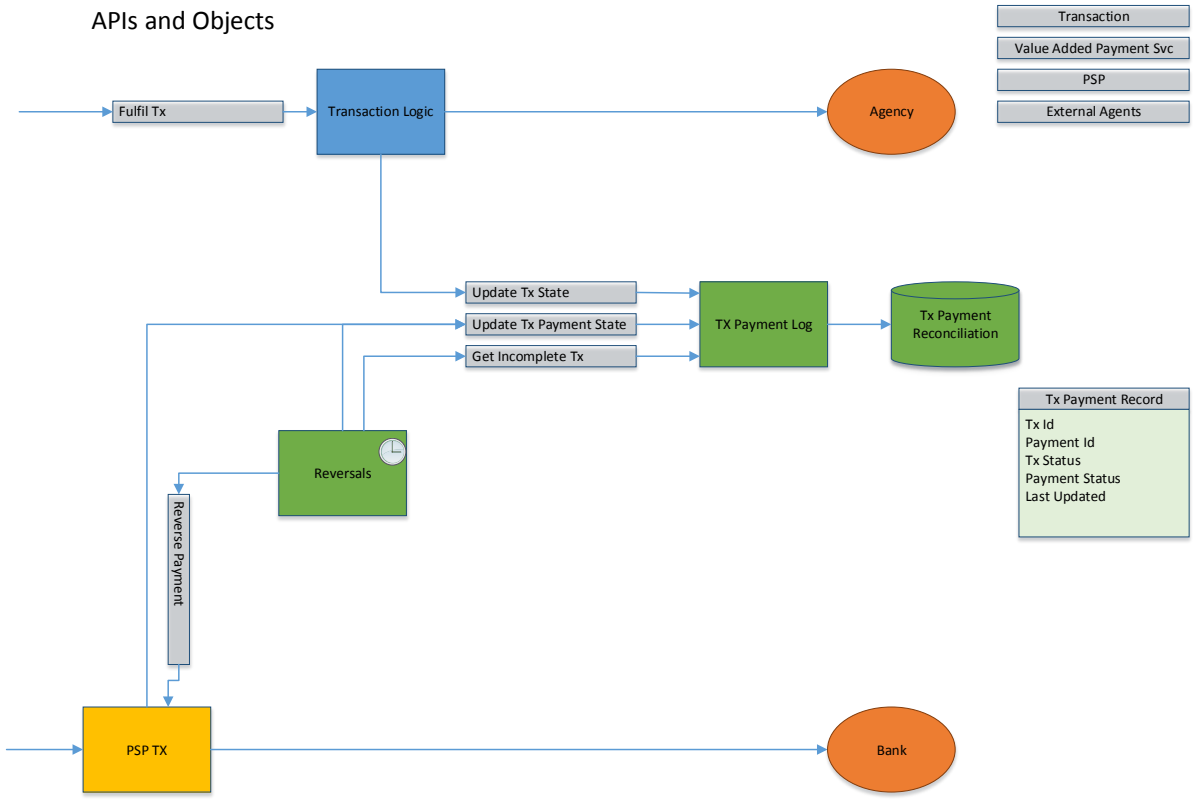
Because the payment is reverse inter day, the payment is reversed with the bank and no impact is made on the user.

Equally because there has been a failure (presumably) with the Transaction UX (such as closing the browser) before the transaction is executed there is no opportunity to leave the customer in an invalid state believing the transaction complete and unpaid.

This solution should simplify significantly reconciliation as only valid paid and processed transactions would now be held by the system and PSP.

4.7.2.3 Implementation and Interfaces

To implement the solution requires the introduction of a new Transaction Payment Log (Value Added Service) and associated Data Store, along with a periodic Reversal component.



The Payment Log should support the following services:

- Update Tx State
- Update Tx Payment State
- Get Incomplete Tx

Update Tx State should allow the Transaction Logic to update the state of a payment transaction, that is update Tx Id, Payment Id, Tx Status, Payment Status, and Last Updated Flag. The intent is that this service is only called from the Transaction Logic component.

Update Tx Payment State should allow the PSP (and Reversal component) to update the state of a paid transaction, that is update the Tx Id, Payment Id, Payment Status and Last Updated Flag.

Get Incomplete Tx should return to the calling system the list of any transactions that are paid but have no Tx Status. That is any record that has a Tx Id, Payment Id, and Payment Status (of Paid), but no Transaction Status.

The Reversals component should execute periodically inter day (say hourly) and query the Payment Log for any incomplete transactions. Where a transaction remains incomplete over an extended period (say an hour) it is deemed to be in failure and reversed through a call to the PSP Reverse Payment service, and the status of the reversal should be recorded in the Payment Log.

Thus hourly a process will run that will reverse any outstanding payments that are more than an hour old. This will address the issue of a payment being made and the transaction not being fulfilled.

4.7.2.3.1 Alerts

When reversing transactions consideration needs to be given to failure states. If (for any reason) there is a failure in the prime transaction processing flow (say agency system failure) it may be

undesirable to reverse a significant number of transactions, simply because the agency system is down. Thus limits need to be introduced to the system.

The System should not reverse payment where:

1. There are over 10 records requiring reversal.
2. There are over $1000 worth of payments requiring reversal.

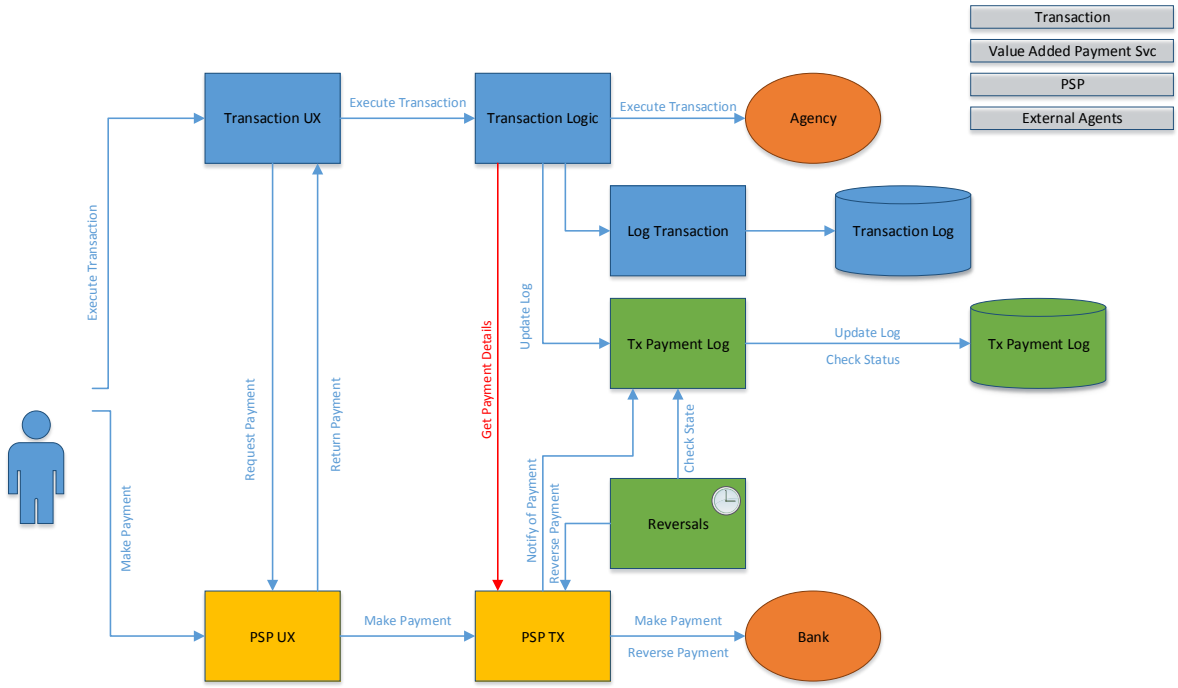
Note: the exact value above can be adjusted.

In both scenarios an email should be generated (by the Reversal component) to Support warning of a potential issue, and requiring manual intervention for reversal.

4.7.2.4 Fraud Prevention

When executing a transaction that involves a payment, a limited set of information is passed from the UX to the Transaction Logic component. The Transaction Logic Component needs to verify that the information passed from the UX is the same as that taken by the PSP. That is someone isn't trying to execute Transaction 27 with a cheaper payment from Transaction 16 (for example).

The Transaction Logic call the PSP service "Get Payment Details" and passes in the Transaction Id, and Payment Id, and if they are a matched pair the PSP will return the Payment Details (including the value paid). If the value paid matches the transaction amount required then the transaction will be processed.



4.7.3 Sensitive Information Handling

Certain data processed, and held, by Service NSW is sensitive in nature, and should be treated with the utmost care and concern.

Security policies are in place for handling sensitive information which must be adhered to.

In general though no Customer or Payment information should be held in a public facing component, application or data store. Further there are few requirements where any form of SNSW information needs to be held in a public facing component. This should be kept in mind when developing solutions.

For initial platform deployments, during HyperCare there is an advantage in not encrypting sensitive data to allow for monitoring and support. However this should never relax data handing to the point that data can be compromised or made public. Once a program exists HyperCare the Personal and Sensitive information should be encrypted and access removed.

4.7.3.1 Personal Data

Personal Data includes any kind of information that allows the public identification of the Customer, such as:

- Name,
- Address,
- Phone,
- Email,
- DOB,

Public exposure of this sort of information would be extremely detrimental to the Customer, and a significant failing for Service NSW.

This information needs to be encrypted in all Logs and Database Records, and should never be held unencrypted in other records. Where this sort of information is sourced from a downstream system, such as SalesForce or and Agency system, the data should be managed by the source system, and not stored by the Transaction Logic processing component.

Personal Data should always be encrypted in transit.

When this information is written to a log, such as is typical of a JSON record. In this situation the data should be encrypted when written to the log:

Original Data:


```
{
  "customerDetails":{
    "customerId":"OLG_CAR_0010341",
    "firstName":"John",
    "lastName":"Smith",
    "birthDate":"1970-11-01",
    "email":"john.smith@hotmail.com",
    "title":"mr",
    "pensioner":false,
    "address":{
      "addressLine1":"10 Park Rd",
      "addressLine2":"",
      "postCode":"2210",
      "state":"NSW",
      "suburb":"PEAKHURST"
    }
  },
  "paymentDetails":{
    "pspReceiptId":"2QwR2v422436659",
    "feeAmount":"207"
  },
  "petDetails":{
    "microchipNumber":"900079000153309"
  },

    "transactionId":"935618",
```

```
    "transactionType":"SNSW-Registerpet",
    "transactionDate":"2018-11-17T00:42:47.283Z"
  }
}
```


Encrypted Data:


```
{
  "customerDetails":{
    "customerId":"CDS431f314f",
    "firstName":"d435df231",
    "lastName":"3421341324",
    "birthDate":"541543245eda",
    "email":"2453rfadr243r",
    "title":"mr",
    "pensioner":false,
    "address":{
      "addressLine1":"32142dswq41",
      "addressLine2":"3124asfqwcs",
      "postCode":"2210",

      "suburb":"PEAKHURST"
    }
  },
  "paymentDetails":{
    "pspReceiptId":"2QwR2v422436659",
    "feeAmount":"207"
  },
  "petDetails":{
    "microchipNumber":"900079000153309"
  },
  "transactionRef":{
    "transactionId":"935618",
    "transactionType":"SNSW-Registerpet",
    "transactionDate":"2018-11-17T00:42:47.283Z"
  }
}
```


Note as the data within the payload is JSON and so it a String, this data can be encrypted in place while leaving the overall message structure. This allows Business Intelligence and Analytics to take place on the overall data structure and meta data, such as Suburb, or Time.

Access to Customer Data should only ever occur via a Value Added Service, that way access to the data is via an API which can be controlled, and monitored.

Encryption should be based on a two part key. The first part should be generic to Service NSW, so that a reasonable minimum key is used. The second part should be specific to the Customer who owns the data. The net result is that every customers data should be uniquely encrypted to that Customer and only decryptable by knowing both the SNSW Key and how the Customer Key was generated.

4.7.3.2 Image Handling

One of the future designs is to hold and reuse the Customers Image. While some Customer Images are specific to the transaction (such as RMS) many situations exist where any (reasonable) image of the Customer could be used. In this situation it is advantageous to the Customer to reuse a “Stock” image of them, taken once, and then reused across various common (non sensitive) transactions.

To achieve this a repository of Customer Images is required, where all image data held in the repository is accessible only via APIs (which therefor can be monitored and tracked), and where all image data held in the repository is encrypted to the Customer.

For an image to be used the Image must first be certified by a trusted source, typically in the initial instance a Service NSW staff member (a CSR), who takes the photo and verifies that the photo is of the Customer. Thus making the photo available for reuse elsewhere.

Encryption should be based on a two part key. The first part should be generic to Service NSW, so that a reasonable minimum key is used. The second part should be specific to the Customer who owns the data. The net result is that every customers data should be uniquely encrypted to that Customer and only decryptable by knowing both the SNSW Key and how the Customer Key was generated.

4.7.3.3 Customer Signature Handling

One of the future designs is to hold and reuse the Customers Signature Image. While some Customer Signature Images are specific to the transaction (such as RMS) many situations exist where any (reasonable) signature image of the Customer could be used. In this situation it is advantageous to the Customer to reuse a "Stock" signature image, taken once, and then reused across various common (non sensitive) transactions.

To achieve this a repository of Customer Signature Images is required, where all image data held in the repository is accessible only via APIs (which therefor can be monitored and tracked), and where all image data held in the repository is encrypted to the Customer.

For a Signature Image to be used the Signature Image must first be certified by a trusted source, typically in the initial instance a Service NSW staff member (a CSR), who takes the photo and verifies that the photo is of the Customer's Signature. Thus making the signature image available for reuse elsewhere.

Encryption should be based on a two part key. The first part should be generic to Service NSW, so that a reasonable minimum key is used. The second part should be specific to the Customer who owns the data. The net result is that every customers data should be uniquely encrypted to that Customer and only decryptable by knowing both the SNSW Key and how the Customer Key was generated.

4.7.3.4 Encryption/Decryption

Encryption and Decryption of data should be provided by a Value Added Service to encapsulate the encryption process, and allow logging and monitoring of actions against the underlying data.

The Encryption Algorithm should be expected to change over time to meet the security needs of the day. The algorithm should always take into account three factors:

- Strength of the Algorithm
- Speed of Execution
- Protection and Generation of Keys

While the strength of the overall algorithm is important in protecting the underlying data, the algorithm needs to operate in a real world environment, quite often in conjunction with real time transactions. So algorithms with intensive long running processing cycles should be avoided.

AES-256 should be used for a default encryption standard. It has proved to be a tested and robust algorithm, with programmatic support in most implementation languages.

The strength and generation of encryption keys determines how vulnerable any encrypted data is to brute force decryption attacks if there is a data breach. That is if a malicious party does obtain a

copy of the underlying encrypted data, how easy is it for them to break the data, and also how much data do they obtain access to by any brute force hack. In simple terms the way to deter this sort of attack is to encrypt different records with different keys. Ultimately a different key per record. Thus forcing the brute force attack to be carried out every time for every record. The net effect is that a large data breach becomes computationally expensive and therefore unlikely to be achieved.

Generating a new key per record is impractical if each key needs to be maintained in an external key store (for example). A suggested alternative is to use a single key (or limited set of keys) to ensure a minimum complexity of key is involved, and then augment this key for each record (based on data not held in the records metadata). For example the Transaction Name, Customer Name and DOB. Thus in simple terms:

- Encryption Key = Service NSW Key + Customer Data
    - Service NSW Key = A21dg34@34d65
    - Customer Data = Pet Registry John Smith 01/02/1970
- Encryption Key = A21dg34@34d65 + Pet Registry John Smith 01/02/1970

Now this still carries a weakness in that once a brute force attack determines the first key the information in the key identifies what is required in subsequent keys (ie Transaction, Name and DOB) to remediate this the Customer Data should also first be encrypted with a different Service NSW Key to obscure it. Thus becoming:

- Step 1 – Generate Encrypted Customer Data
    - Service NSW Key 1 = adfa324dca&6231cf
    - Customer Data = Pet Registry John Smith 01/02/1970
    - Encrypted Customer Data = gfd3241da314@#hcf

- Step 2 – Generate Encryption Key
    - Encryption Key = Service NSW Key 2 + Encrypted Customer Data
        - Service NSW Key 2 = A21dg34@34d65
        - Encrypted Customer Data = gfd3241da314@#hcf
    - Encryption Key = A21dg34@34d65gfd3241da314@#hcf

With this approach even if a brute force attack is used, there is no visible evidence of the Customer Data used to generate the key. However it is still possible to identify at least the common Service NSW key component as this would be common across all keys. Thus a third step is required to obscure the combined key. Thus becoming:

- Step 1 – Generate Encrypted Customer Data
    - Service NSW Key 1 = adfa324dca&6231cf
    - Customer Data = Pet Registry John Smith 01/02/1970
    - Encrypted Customer Data = gfd3241da314@#hcf

- Step 2 – Combined Encryption Key
    - Combined Encryption Key = Service NSW Key 2 + Encrypted Customer Data
        - Service NSW Key 2 = A21dg34@34d65
        - Encrypted Customer Data = gfd3241da314@#hcf
    - Combined Encryption Key = A21dg34@34d65 + gfd3241da314@#hcf

- Step 3 – Obscure Encryption Key

- Combined Encryption Key = A21dg34@34d65 + gfd3241da314@#hcf
- Service NSW Key 3 = da#54dfa3213
- Obscured Encryption Key = ad%7h5^5gfdsal21ncJnLKBsha2341

This process renders the final encryption key to be meaningless across all brute force hacks on any record, making access to the underlying data being protected of limited value.

The above is provided as an example of the mechanism only, and can be greatly simplified during implementation. It does however demonstrate a way forward where every record for a Customer is encrypted separately based on the Transaction Type, Customer Details, and an underlying SNSW Key, without the need to store and manage a massive number of individual keys.

By using a Value Added Service the Encryption Mechanism, particularly the method of key generation, can be completely obscured from the majority of developers, yet still be made available to those developers through a controlled programmatic API.

4.7.4 Agency Availability, performance and Caching

When dealing with large agencies, there is a reasonable expectation that the agency system is highly available, highly resilient, and highly performant.

When dealing with smaller agencies, or where IT is not their primary focus, it is not reasonable to expect the same level of availability or performance from their systems.

Our Customers and Staff expect the same level of availability and performance across all systems presented by Service NSW. This is not an unrealistic expectation as the Customer has no knowledge of the underlying agency, their capabilities or priorities.

This presents a problem for Service NSW. Uplift Agency Capability is time consuming and costly, and typically a cost born by Service NSW. While it is almost always preferable to uplift the Agency Capability it is not always practical.

Caching can be introduced in this situation to a Transaction through the use of a Value Added Service, a Caching VAS. The Caching VAS should allow for temporary storage of data from the downstream agency, to allow the smooth provision of Customer Capability.

It should be noted that Caching can not solve all Agency Provisioning issues, and is not suitable to all transactions. It should be applied on a Transaction by Transaction basis to suit the needs of the transaction. In all situations the Cached data should be short lived (at most 24 hrs, in most situations minutes is preferable).

The Cache VAS should offer simple APIs:

- Write to Cache
- Read from Cache
- Delete from Cache

All data stored in the Cache should be encrypted. For caching the encryption required is not as significant as that used for long term data storage as the data is short lived.

4.7.5 Large File Handling

The current implementation of the Transaction Logic Components utilise the APIs and APIGEE Service Bus for the passing of Data Files (PDF Files, Image Files etc). This has the advantage of not requiring the handling or storage of this data outside the SNSW environment, thus reducing the risk

of data loss, while enabling the underlying transmission protocols to handle data encryption and protection.

It is noted that as technology changes, these files may increase in size, particularly image files, which may then impact the performance of the Service Bus platforms. The risk of this impact on the Service Bus has been anticipated, and mitigated, through the reduction of processing in the Service Bus. Still the risk exists, and may need to be addressed in future.

One solution is to use external storage for the handling of file data, and therefore only pass references to the file data through the Service Bus. This has the advantage of not carrying the load of a files raw data through the Service Bus. This has the disadvantage of creating a point of weakness, where files are now stored outside the SNSW environment, and now need to be managed.

To date the impact of processing file data through the Service Bus has not justified the complexity of managing file data outside the SNSW ecosystem.

4.7.6 Data Storage

The current architecture design is based on a Stateless Microservice Architecture, and so has little to no Data Storage demands (outside that required for Audit Logging and Metrics Logging).

A Stateless Microservice Architecture may not ultimately be applicable to all Transactions. Some Transactions may require a local operational data store (note this should be the exception, not the norm). The use of a local operational data store is problematic for PCF components as PCF has no concept of localised File Storage.

Therefore the suggested approach for Operational Data Store for transaction data should be the use of a localised AWS database provisioned via PCF. This database should be used only by the transaction it is intended for, and not store data for an extended duration.

- No data should be kept for more than 30 days.
- All personal data should be encrypted.

4.7.7 Alerts

Alerts need to be built into the underlying architecture through a Value Added Service, so that when a transaction (or supporting component) fails for any reason an alert can be logged, and then (where necessary) raised to a support operator for investigation.

Currently the process of identification of failure relies greatly on after the event resolution (ie notification from a Customer or CSR of failure). This makes the response reactive in nature, and places the supporting teams under considerable pressure as the failure has already been noticed in production.

A more sustainable solution is to log the Alert through a Value Added Service, and when a threshold is reached (for an alert type) an SMS or Email is automatically sent notifying support of the need for further investigation.

This approach should be built into all Transaction Components, and used whenever failures or exceptions occur.

4.7.8 Digital Asset Management

With the UX Components being light weight throwaway implementations, there is a need to provide a level of consistency and control across what is displayed and how. To achieve even a basic level of consistency a Digital Asset Management (DAM) platform is utilised.

The DAM is implemented as a lightweight CMS platform, provisioned on AWS, using a WordPress Instance.

WordPress was selected not so much for its ability to deliver CMS web pages, but rather because of its ability to offer API access to the underlying content stored in its repository. No actual web pages are delivered through WordPress (a separate Digital Drupal instance is used for this, outside the scope of this paper). WordPress is purely used to manage content snippets, images, copy text, and other static content that is utilised by the UX Components.

In general the pattern followed is for the UX Component, on start up, to access the static content it required to generate a page from the WP DAM. This content is then rendered onto the Clients screen.

The advantage of hosting the Static Content outside of the core application component is that the content can be changed independently of the software release cycle. The Business can change copy text any time regardless of the phase of a software release. Note this is an ideal situation which is rarely achieved, but a worthwhile aim.

The level of content stored in the DAM will vary depending on the Transaction being accessed. In general any static content that is likely to change, such as welcome messages, headers, footers, help text, survey questions, etc should be stored in the DAM.

The DAM provides a simple editor, and version control allowing for easy editing and publishing of content.

The DAM has its own internal cache, which requires manual refreshing when changes are made (for immediate reflection in the UX), though there is a daily refresh of the WP Cache.

The DAM also sits behind the AWS CloudFront CDN, another form of a cache, a faster cache, which must also be refreshed for any immediate change to be reflected in the UX Component.

That is there are two caches which must be refreshed for an immediate content change.

Note the WP Cache is now somewhat redundant through the use of the CloudFront CDN, but has not yet been removed from the implementation.

4.7.9 CRM

For the foreseeable future the CRM of choice will remain SalesForce.

SalesForce will continue to be used for the management of Customer Data, and Case Records.

Where Customer Data is required in a Transaction process this should be sourced from SalesForce.

Where Case Data is required to be held, this should be written back to SalesForce.

The interface to manipulate information within the CRM should be centralised through a Value Added Service (one VAS for Customer and one VAS for Case).

The Customer VAS should allow the manipulation (Find, Create, Read, Update, Delete) of all Customer Data.

The Case VAS should allow the manipulation (Find, Create, Read, Update, Delete) of all Case Data.

The Case VAS should also allow the manipulation of a Cases State, and manipulation of a Cases Queue Mechanism (placement and movement across queues).

4.7.10 Dashboards

Much of the Architecture focuses on the delivery of Transactions. There is also the need for cross transaction capability. One example of this is the idea of a Dashboard, a central work space (or portal) allowing access by a user to the transactions offered by SNSW, and historical information related to these transactions.

The Dashboard can be seen a new UX Component, that provides a menu/tile style layout to access Transactions offered by the platform. Each transaction operates in its own right through its own public interface. The Dashboard simply presents a common centralised location for access and management of the discrete transactions.

The exact nature and function of a dashboard will be dependent on the underlying capability available.

5 Implementation Patterns

This section demonstrates the basic Omni Channel Reference Architecture, and shows how this can be extended to cover additional capabilities.

5.1 Single Channel Implementation

The single channel implementation is the minimum implementation pattern that demonstrates the Reference Architecture.

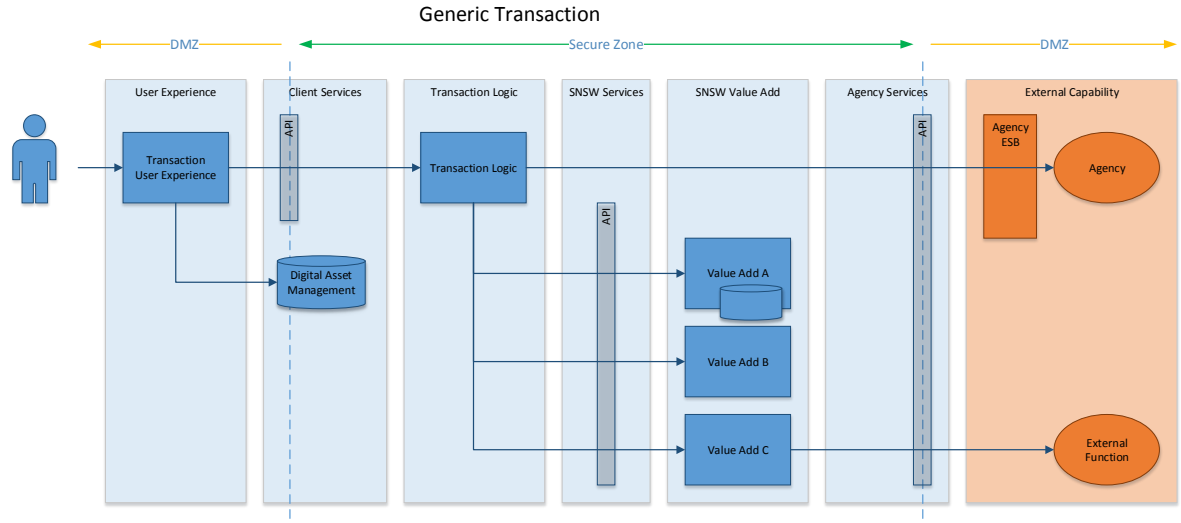
It utilises a single User Experience Component, a single Transaction Logic Component, leverages a number of Value Added Services, and integrates to the downstream agency.

In this model there is also depicted the use of the DAM, and integration to an external service via a Value Added Service.

This model does not depict the use of Security, Payments, or CRM.

With this model care must be taken not to build Business Logic into the UX Component as future iterations may involve the delivery of a Multi Channel Solution.

With this model care must also be taken to extract common static content to the DAM, so that changes are not required when adding additional channels.



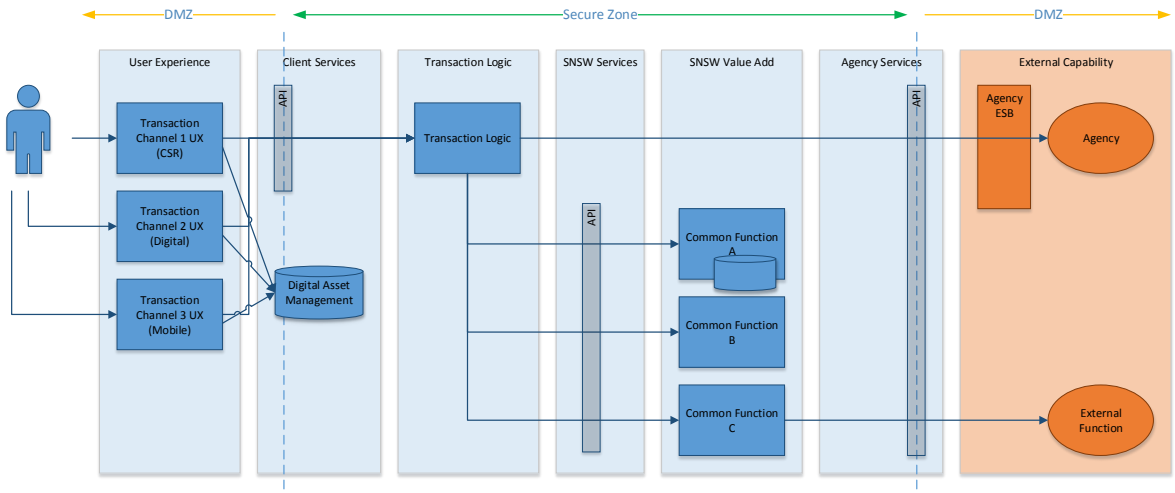
5.2 Multi Channel Implementation

The Multi Channel Implementation extends the Single Channel Implementation through the addition of Multiple UX Components.

The Transaction Logic would remain extensively identical to the single channel model.

Additional Channels are added by the development of new UX Components specific to the channel desired.

Multi Channel Transaction with DAM



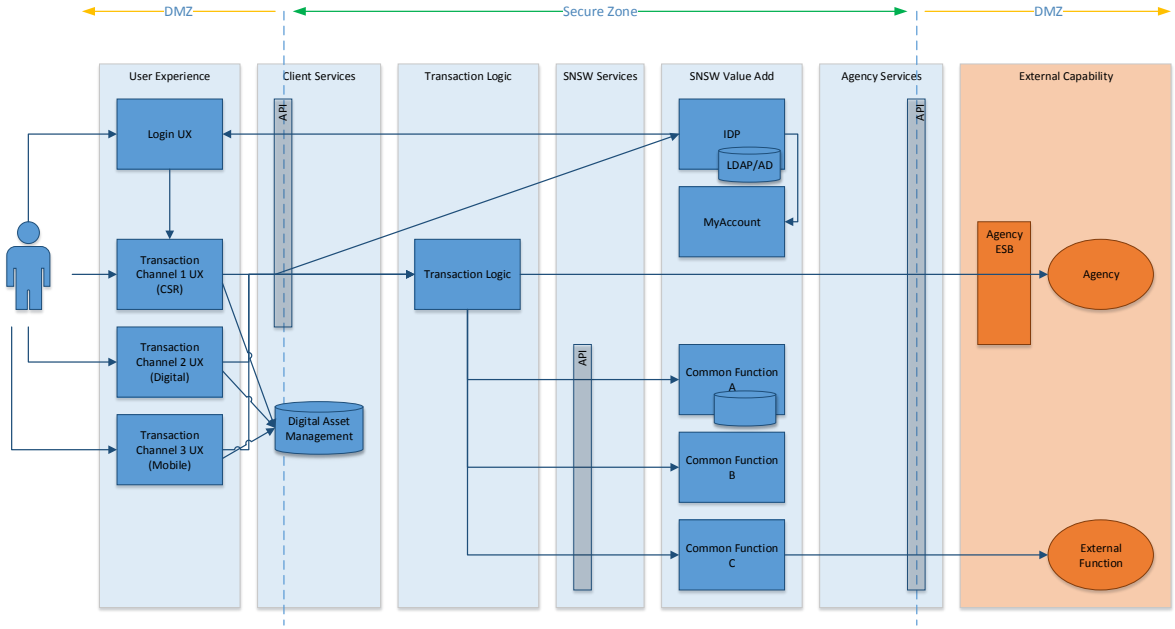
5.3 Secure Multi Channel Implementation

Security is applied to the model through the use of the Ping Federate IDP component, and MyAccount.

Security is applied at the API layer, requiring the User to Authentication prior to accessing and executing specific APIs. When configured, the API layer will verify the users credential (Access Token) against IDP, prior to allowing access to the API.

As security is applied at the API layer the security must be initiated within the UX Component. The UX Component Authenticates the user via the IDP, which in turn requests the user to login. Where additional information is required, such as linking to the agency, this is enabled through the use of MyAccount.

Secure Multi Channel Transaction with DAM

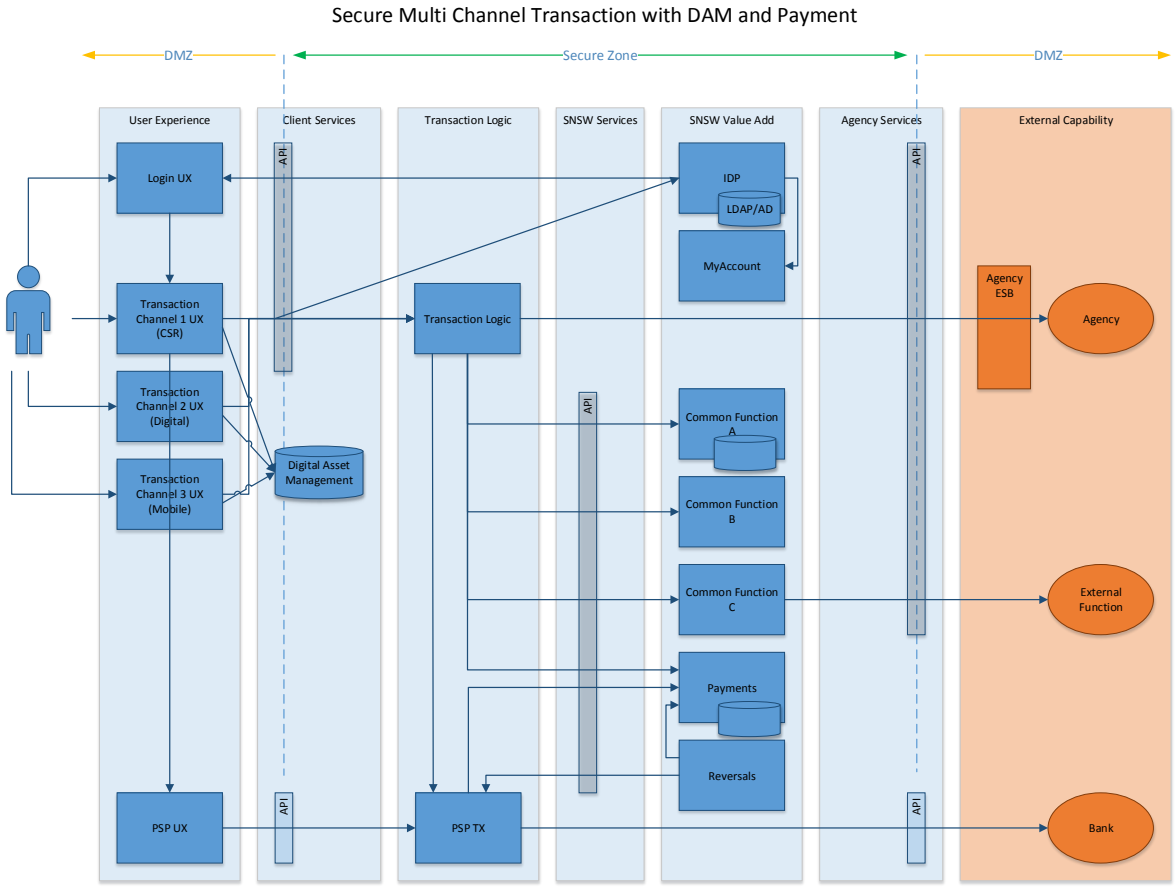


5.4 Secure Multi Channel Implementation with Payments

Adding payments to a transaction involves the redirection of the UX Component to the PSP UX, and the re-instantiation of the original UX Component on completion of the payment.

Additionally there is back end integration between the Transaction Logic and the PSP Tx Layer to allow the validation of the payment, and where necessary the reversal of the payment.

There is also the addition of new VAS components to track the payment and handle the reversal of the payment if the transaction fails.



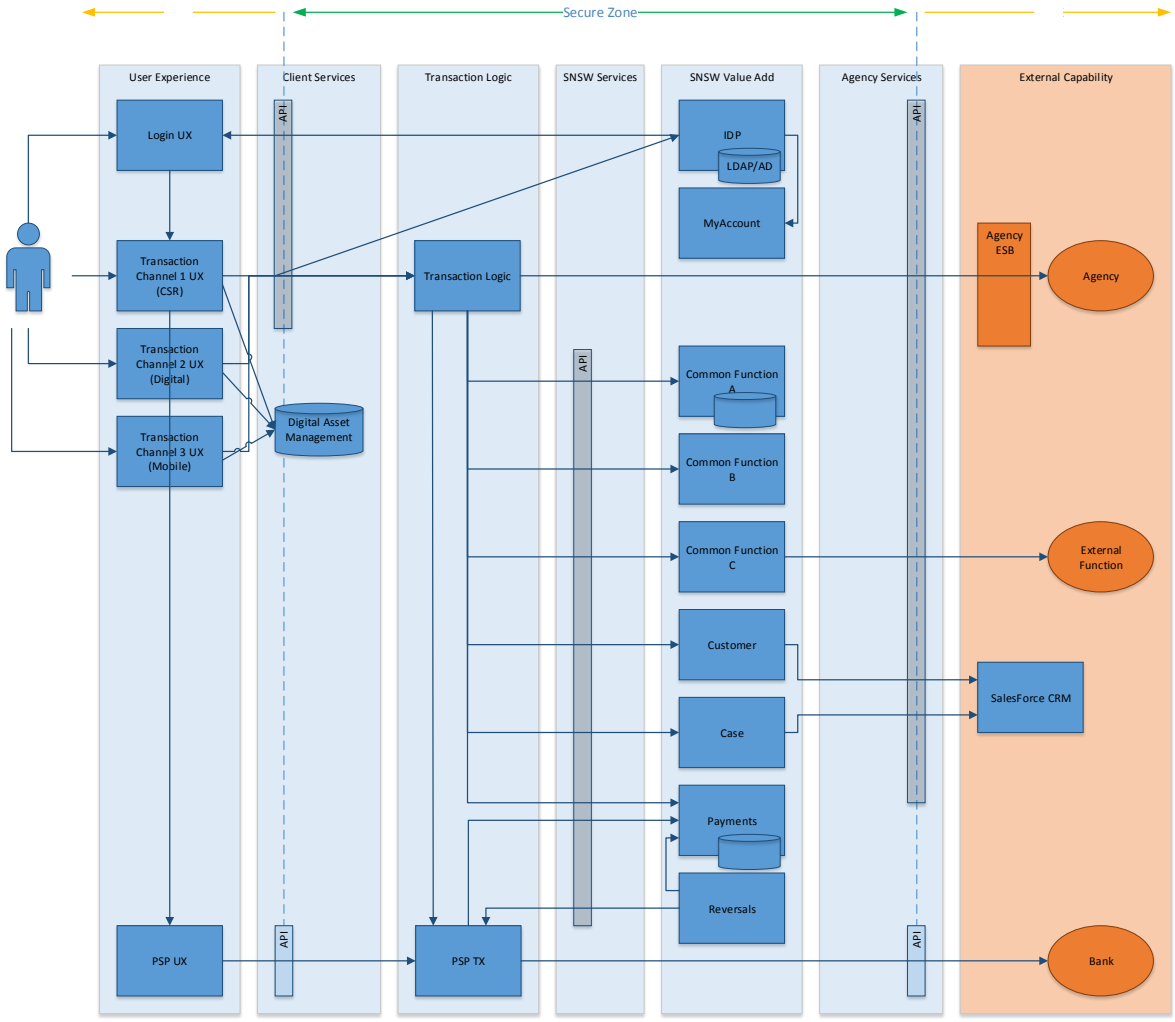
5.5 Secure Multi Channel Implementation with Payments and CRM Integration

CRM integration allows the Transaction Logic Component to interact with the back end SalesForce CRM platform.

Integration to the CRM platform is via two separate Value Added Services, one for Customer Details, the other for Case Details.

Note while in this diagram Customer Details are sourced from the CRM, this model also allows the Customer VAS to source Customer Details from multiple sources, thus providing an element of a single view of Customer.

Secure Multi Channel Transaction with DAM, Payment, and CRM

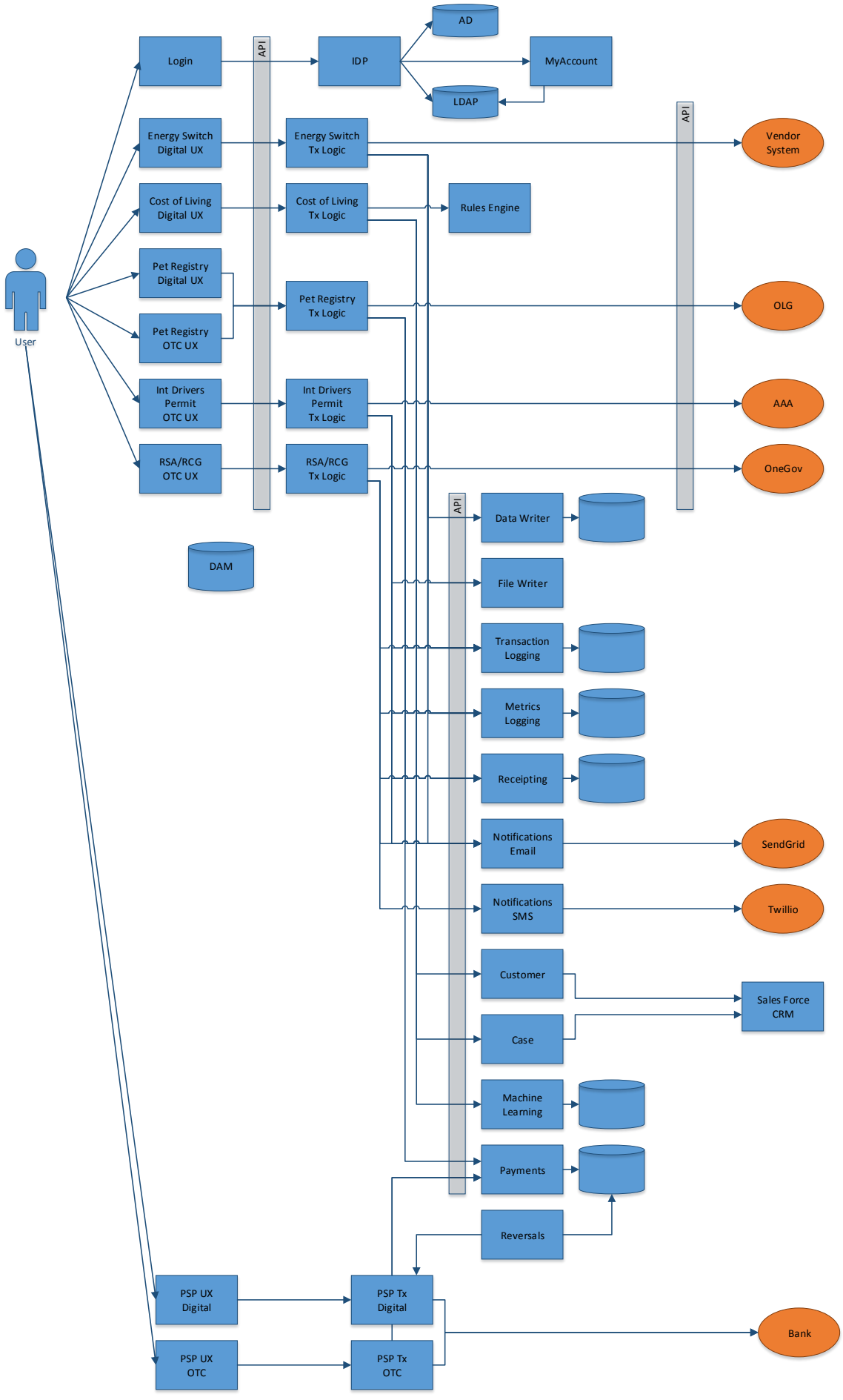


5.6 Full Environment

The entire environment can therefore be depicted across a number of transactions as shown:

- Energy Switch
- Cost of Living
- Pet Registry
- International Drivers Permit
- RSA/RCG

All of which utilise and share a number of Value Added Services, Payments and Security.


AD
Login
API
IDP
MyAccount
LDAP
API
Energy Switch Digital UX
Energy Switch Tx Logic
Vendor System
Cost of Living Digital UX
Cost of Living Tx Logic
Rules Engine
User
Pet Registry Digital UX
Pet Registry Tx Logic
OLG
Pet Registry OTC UX
Int Drivers Permit OTC UX
Int Drivers Permit Tx Logic
AAA
RSA/RCG OTC UX
RSA/RCG Tx Logic
OneGov
API
Data Writer
DAM
File Writer
Transaction Logging
Metrics Logging
Receipting
Notifications Email
SendGrid
Notifications SMS
Twillio
Customer
Sales Force CRM
Case
Machine Learning
Payments
Reversals
PSP UX Digital
PSP Tx Digital
Bank
PSP UX OTC
PSP Tx OTC

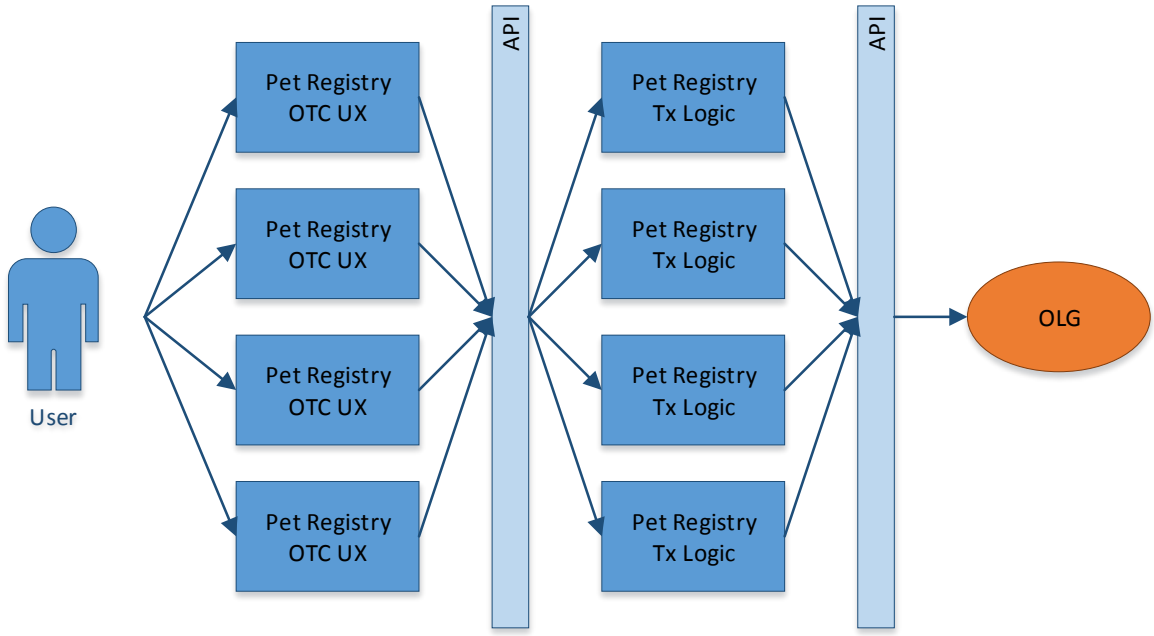
6 Future Direction

6.1 Fault Tolerance and Redundancy

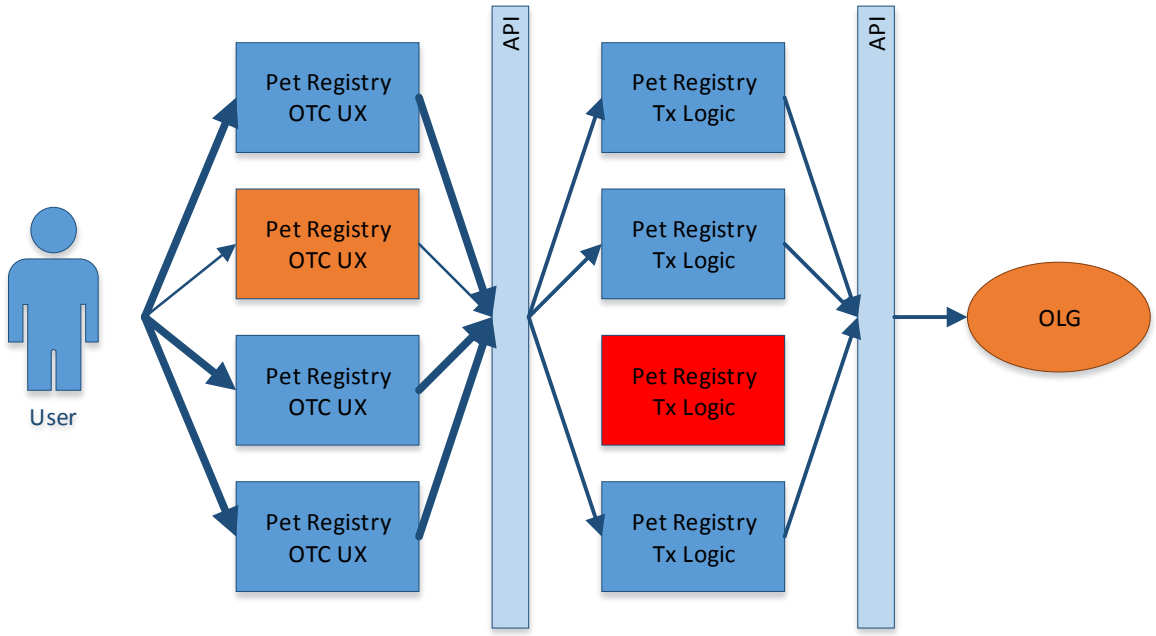
The Omni Channel Reference Architecture provides a highly componentised solution, with little to no dependency between transactions. Each transaction in this model is now encapsulated into its own User Experience and Transaction Logic components.

When transactions are independently encapsulated faults within a component are limited in impact to that specific component. By allowing the split between the UX and the Tx Logic there is independence of Form and Function. Scaling allows for the duplication of components and in effect the delivery of a network capable of now sustaining an outage in any single component.

A simplistic example of a scaled network using the microservice topology is shown.



If a Component in the network starts to under perform, or a component simply fails, the network can simply rout around the failed components and continue operation.



At this point a poorly performing or failed component can simply be restarted and brought back into the network.

There are with such a design critical platforms, components that represent a single point of failure. In the above example the API layer represents such a point of failure, as does the downstream agency OLG.

These single points of failure are a risk to Service NSW and need to be investigated and strengthened so that they do not have a detrimental impact on the platform.

The key areas of risk for such single failure points are:

- API Layer
- IDP Security Layer
- PSP Payment Platform
- SalesForce CRM
- Downstream Agencies
- Digital Asset Management Platform
- Databases

While some redundancy can be built into each of these platforms, it is not the intent of this paper to provide solutions for them. Further investigation is required.

6.2 Scaling

With the use of a Component model, and a microservice architecture it is necessary to tune the solution for optimum throughput. Tuning involves scaling components within the network.

Any component within the network can run as multiple instances. With only a few exceptions (such as batch processes) most components should be set with a minimum number of instances of 2. Thus there is always a distribution of load and a failover in case of errors.

The upper limit of scaling is determined by the capacity of the PCF environment, and the resources (typically memory) consumed by any component.

In theory the PCF environment is capable of scaling the number of component instances to match the load. With the designer only having to set limits (such as if capacity > 70% then add more instances).

In reality the PCF environment has a bug and does not scale. Thus it is important for the delivery teams to correctly tune their applications and the number of components to handle the anticipated load.

While there are no set guidelines (as transaction processing and throughput can vary greatly), observation has seen that most components only need between 2-6 instances to handle the sort of load experienced by Service NSW.

6.2.1 Scaling VAS Components

VAS represent a special circumstance.

While there are over time only expected to be a limited number of different VAS Components, these components are going to be used across many different transactions. Thus a common VAS component (such as Logging) will take significantly greater load than any single TX Logic Component.

These components therefore need to be scaled to a higher level than other components to handle the cross transactional load.

6.3 Databases

The current Database implementation leverages AWS RDS and utilises a Microsoft SQL Server DB.

Currently there is no requirement for a specific DB Implementation, Microsoft SQL Server was selected simply because of its ease of use, knowledge within the team, simplicity of operation, and compatibility with development platforms.

The only DB specific technique used by the team is the use of SEQUENCES in the SQL Svr DB. The generation of Sequences is used across the microservice architecture to allow the unique generation of Transaction Numbers without the need to revert to a more complex GUID style arrangement.

No tuning has been conducted on the DB. It has simply been used as a set and forget datastore.

The current DB implementation is an Enterprise Standard deployment (multiple cores, and failover).

Further investigation may be warranted for a corporate standard DB.

Regardless of which DB technology is used, efforts should be made to limit the use of platform specific DB techniques (such as Stored Procedures).

7 Example Implementations

Presented here are the architecture layouts for the first four transactions utilising the Omni Channel Reference Architecture.

These demonstrate the reuse of Value Added Components, the delivery of new capability, and the clear segregation of components.

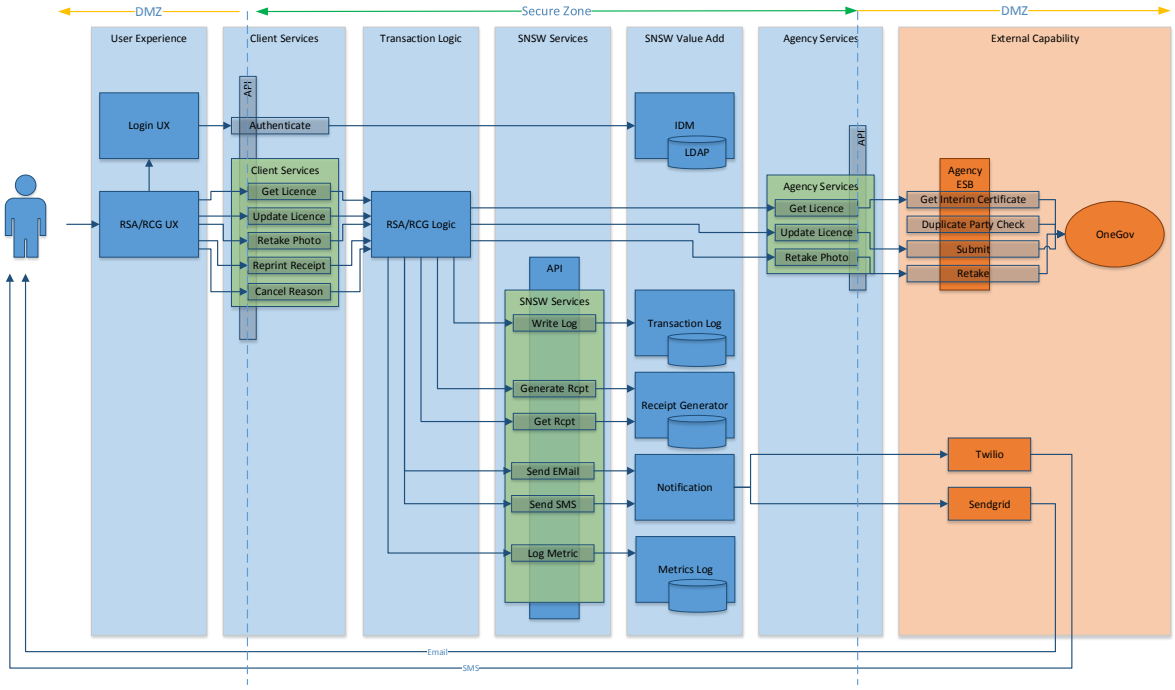
The aim of these examples is to provide guidance as to how new transactions can be integrated into the Omni Channel Reference Architecture.

7.1 RSA/RCG

The RSA/RCG transaction integrates to the OneGov GLS platform to allow Customers to register new RSA/RCG licences.

The transaction is offered only over the counter by A CSR, as it requires POI, and Image/Signature capture. It implements Security, Receipting, and Notifications.

RSA/RCG High Level Architecture



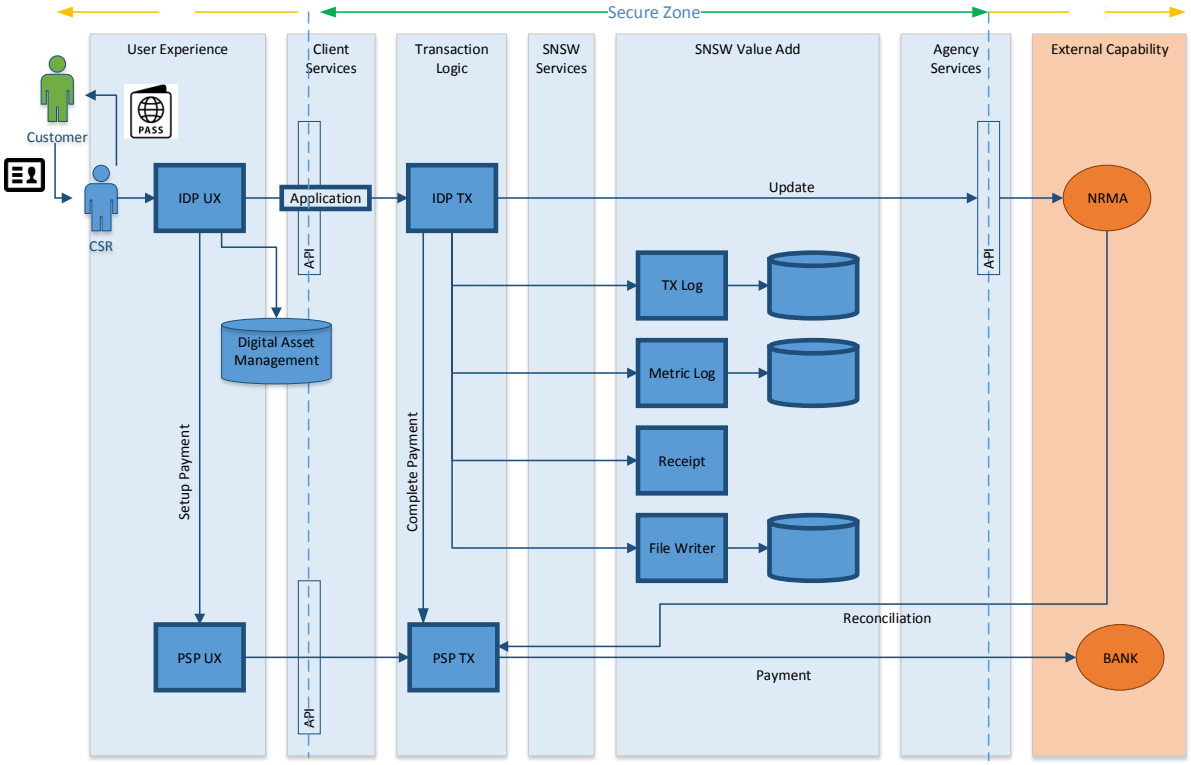
7.2 IDP

The IDP transaction utilises a batch file integration to a downstream public enterprise (NRMA). The transaction enables Customers to purchase an International Drivers Permit from a CSR.

This is the first transaction to utilise the PSP payment platform.

This transaction also introduced the use of the DAM.

IDP High Level Architecture



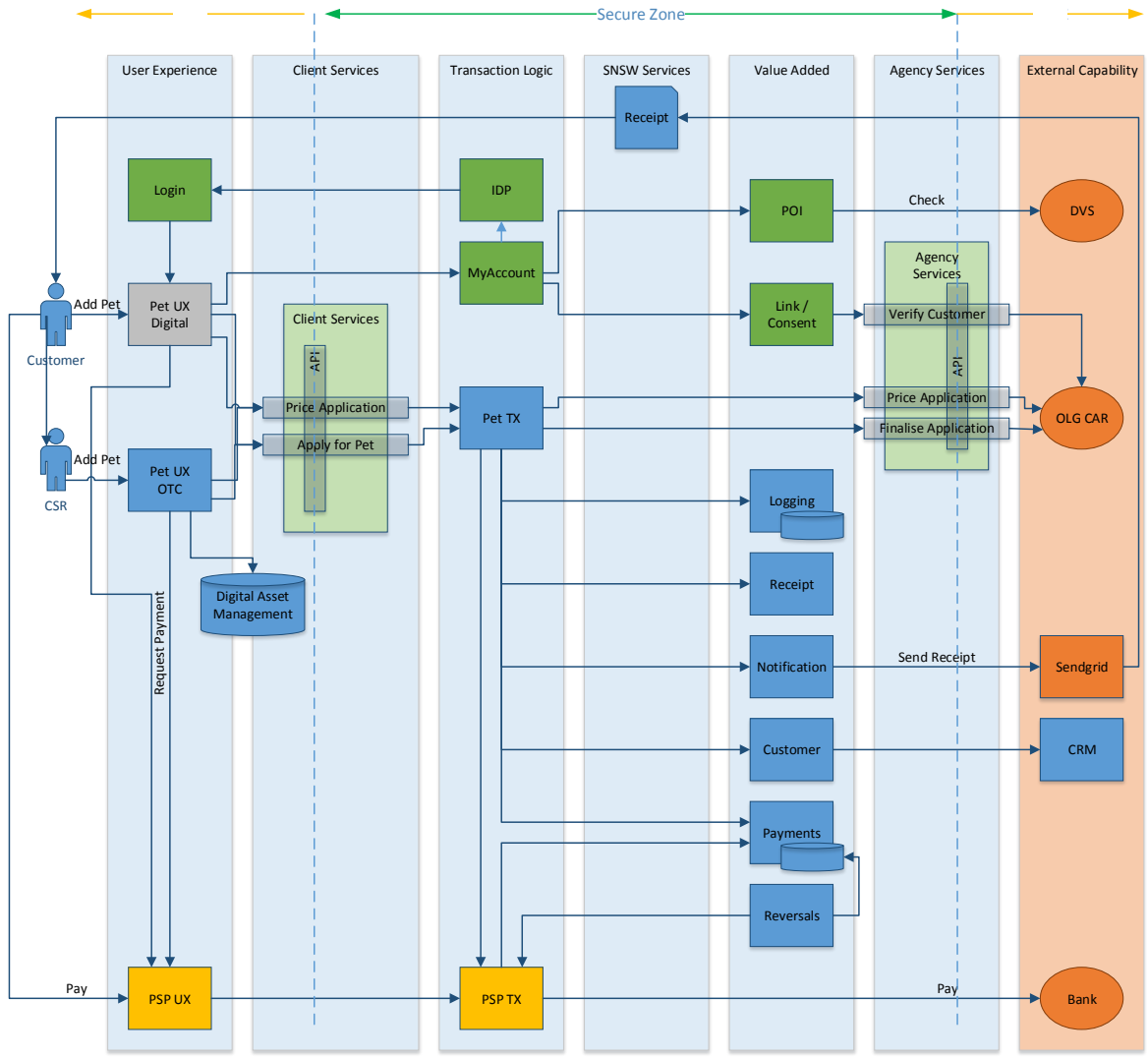
7.3 CAR

The Companion Animals Register transaction is the first to integrate a new government agency (OLG), and also the first to offer multiple channels for transaction (both Digital and Over the Counter).

The platform fully integrates to MyAccount, and the PSP, and implements the reversal process mimicking a two phase commit for payments.

It is also the first transaction to integrate to SalesForce CRM and develop out the Customer VAS.

CAR High Level Architecture



7.4 Cost of Living

The Cost of Living program is a Digital Transaction wholly contained within Service NSW (no downstream agency integration).

The transaction leverages MyAccount, and introduces new capability in terms of a Rules Engine, Machine Learning Data Store, and Case Integration to SalesForce.

This transaction also integrates to the DAM.

Cost of Living High Level Architecture