# NEC Australia NFW Campaign: Clubs New South Wales 2023-03

* Vendor endorsement: Clubs NSW has stated that they cannot endorse a single vendor and push the same FR vendor onto all smaller gaming clubs. This means that each club will have to make their own decision as to which vendor they want to work with. As a result, we will need to be prepared to pitch NFW product and differentiate NFW from the competition.

* Standard database: Clubs NSW is interested in having a standard database for the self-excluded list. This is important to ensure that all clubs are using the same system and that there is consistency in the data. We should focus on making sure that NFW meets the standards and requirements set by Clubs NSW or alternatively, we can showcase our database format as a benchmark for best practices.

* Head office (Master) database: It sounds like the plan is to have a master database at head office aka Clubs NSW which will receive photos and applications via file transfer. And replication of the database may be problematic as there is no network to the small gaming clubs. We will need to find a solution to this problem to ensure that the database is replicated correctly.

* API: The database at club level needs to be open API for any facial recognition the gaming club chooses. The database needs to be easily accessible and compatible with a variety of different software systems. We can perhaps share the NFW restful API documentation with the other clubs for consideration.

* Master file: It is unclear how a new excluded player gets onto the master file. Do they go to Clubs NSW and get registered, or do they go to an individual gaming club and get excluded? We will need to clarify this process with the customer and ensure that it is easy for individuals to get registered and excluded from all venues if necessary.

# Logo & Branding

A black and white circular logo

Description automatically generatedIt will appear in NFW as a new button with a logo similar (but simpler than) this. The product won’t be called nexus it will be called NeoFace Watch Master List.

# Solution Architecture

The following application and cloud architecture has been tested and validated by Luke Egging working together with CST (Adam Gillet) and IT Security (Rob Rice) on 25th October 2023.

## Application Architecture

A diagram of a computer program

Description automatically generated

Figure 1 - High Level Components

|  |  |
| --- | --- |
| Pros | Cons |
| Don’t need to develop the backend at all  Existing technologies, widely used  Automatic DR and Scale  Easy to control access (Azure IAM)  Easy to secure (App Gateway)  Human readable files (if cleartext XML/JSON). I can compare backend to NFW and quickly see if we’re up to date  We can control limits  We know when recipients read (so we can identify problem-recipients/heavy users) (API Gateway Reporting) | Inefficient storage (minor)  Azure only  Support from Microsoft can be slow due to teams (blob vs appgateway) |

Figure 2 – Pros and Cons

|  |
| --- |
| **Central DB Availability Assumptions**   * Recovery Point Objective RPO (24hrs) * Recovery Time Objective RTO (24hrs) (depending on staff) * Local Redundant Storage LRS   **Simple storage structure, e.g.**   * Single Json Index file contains list of all images including (URI to image, firstname, lastname,Unique ID,eye positionxy,head rect xy w/h, add image version id (e.g. utc datetime), plus a hash of text to help clients detect change) * Images stored raw in /image/\*[.jpg   + Clients determine something has changed when the hash is different to the one they have on file. * (JPG supported only) |

Figure 3 - Core Architecture Principals

## Cloud Architecture

[2023-02 Nexus 2.0 - Australian Casinos.vsdx (sharepoint.com)](https://necsws-my.sharepoint.com/:u:/p/adam_gillett/EdCSJ97Y3kVJgMbnBghD99YB3llBnWN2KDVmpV5ASpDhpA?email=luke.egging%40necsws.com&e=4%3ARHxTDf&fromShare=true&at=9&CID=a592fb0b-3e8e-2d1d-078c-94ba23b410b0)A diagram of a computer network

Description automatically generated

### REST API Client Requirements

Clients may interface with the central repository either through an internet facing READONLY endpoint, or a READWRITE endpoint.

A screenshot of a computer

Description automatically generated

In our testing, this is the kind of information that clients will need to talk to either endpoint

### A computer code on a black background Description automatically generated

# Index Data Structure (Central Repository)

A screenshot of a computer

Description automatically generated

|  |
| --- |
| Deltas folder |
| Sample contents:    Sample Delta: {"WatchlistName":"Problem Gamblers", "Subjects":[{"FirstName":"Peter","LastName":"Smith","SubjectID":"8edd6a91-e596-49f5-a3ae-53099d2172f6","Change":0,"PhotoMd5": "xxd4355","Head":{"x":20,"y":44,"width":50,"height":150},"LeftEye":{"x":40,"y":100},"RightEye":{"x":70,"y":110}, "AdditionalPhotos":[{"Md5":"x5wfs92","Uri":"Photos/107a97e7-0f7c-4f14-bdd5-150d1da70256.jpg","Head":{"x":20,"y":44,"width":50,"height":150},"LeftEye":{"x":40,"y":100},"RightEye":{"x":70,"y":110}}]}]} |
| **Rules**   * Contributors are responsible for adding delta files and processing them. * The last contributor that held the lock should also check and process deltas from other contributors, if they have that role. * In principle, Delta files should not remain for long – they should be applied to the main Index as soon as possible. |

|  |
| --- |
| RejectedDeltas folder |
| Sample contents:  A number on a white background  Description automatically generated |
| **Rules**   * Contributors are responsible for adding delta files and processing them. * RejectedDeltas will remain permanently * RejectedDeltas folder is needed for any Deltas that fail to be applied, e.g. an update to a record that was received AFTER the record had already been deleted. * These should be rare |

|  |
| --- |
| ProcessedDeltas folder |
| Sample contents:  A number on a white background  Description automatically generated |
| **Rules**   * Once contributors successfully process a delta, it is moved here for the record. * Processed Deltas will remain for 180 days (configurable), after which they will be deleted permanently, along with their respective photo. |

|  |
| --- |
| Photos folder |
| Sample contents:  A screenshot of a computer  Description automatically generated |
| **Rules**   * Contributors are responsible for adding/removing Photos. * Photos should not remain when records are deleted |

|  |
| --- |
| Index.json |
| **Sample content (TODO: Update)**  {"WatchlistName":”Problem Gamblers”, "Subjects":[{"FirstName":"arn","LastName":"shc","SubjectID":"8edd6a91-e596-49f5-a3ae-53099d2172f6","LastUpdateTimeUtc":"2023-10-16T18:23:01.428","Change":0,"AdditionalPhotos":[{"UriToImage":"Photos/107a97e7-0f7c-4f14-bdd5-150d1da70256.jpg"}]},{"FirstName":"dan","LastName":"craig","SubjectID":"9c7a97e7-0f7c-4f14-bdd5-150d1da70256","LastUpdateTimeUtc":"2023-10-16T18:24:46.631Z","Change":1 }],"LastUpdateTimeUtc":"2023-10-16T18:27:40.772Z"} |
| **Rules**   * Contributors are responsible for maintaining the index when changes are detected * One JSON object per watchlist * Subject Default photo URI will be inferred as <SubjectID>.jpg (so doesn’t need to be explicitly configured) * AdditionalPhotos can be not present or null (in most cases will not be present as only one photo of the individual is expected on file) * LastUpdateTimeUtc does not need to be a datetime – could be some other hash but we have a proposal for this later. |

|  |
| --- |
| Deleted/Deleted-2023.json |
| **Sample content (TODO: Update)**  {"WatchlistName":null,"DeletedSubjects":[{"SubjectID":"8edd6a91-e596-49f5-a3ae-53099d2172f6", "DeletionTimeUtc":"2023-10-16T18:23:01.428", "DeletionReason": "Expired", "DeletionNotes": ""}], "DeletionReasons": {"Expired", "OperatorDecision", "UserRequest"}} |
| **Rules**   * Contributors are responsible for maintaining the deleted index when changes are detected * New file created each year (done by Contributor workflow) * Permanent record of deletions are stored here * No Personally Identifiable Information, PII, stored here |

# States

The Service will have the following states.

* Idle (waiting for next sync time)
* Delta syncing (only looking for and applying “differences”)
* Performing a full re-index

# Contributor-Specific Requirements

Contributors will have additional states:

* Uploading a delta or deltas
* Getting a lock on the index
* Updating the central index
* Releasing the lock on the index

| **Central NFW System in ‘Contributor mode’**  New or Existing Feature? | | |
| --- | --- | --- |
| Existing | Enrol | Biometric de-duplication check. Prevent duplicate images from being registered. We can do this automatically, or manually by getting users to do a face search before doing an enrolment. |
| New | Watchlist Sync | When enrolling a new subject into NFW, also save the Subject Image(s) and metadata as a JSON index file plus image file into a private, secure Azure blob Storage account.  Initial version: windows service which synchronises a single watchlist based on config.  Future version: create a GUI page in Nexus to configure this. |
| New | Watchlist Sync | When updating a subject in NFW, also save the changes to the subject/metadata into the JSON index file plus image file into the Azure Blob Storage  How to detect all updates from NFW so that we can communicate them?  Initial version: the new service app could use MS SQL “Change Data Capture” feature <https://www.c-sharpcorner.com/UploadFile/db2972/change-data-capture-in-sql-server/> - and we can periodically query the tables for updates. |
| New | Adaptable Central Database Connectors | The Nexus 2.0 Service should be flexible by design, using the “adapter design pattern” to abstract the underlying backend database technology. E.g. runtime DLL Loading (or better method) should be used at service startup, to interface with the appropriate backend database. This design will allow the application to be able to connect to completely different central database products in the future with minimal additional development effort.  The initial application will use Azure Blob Storage as the central repository.  This SHOULD (not must) also work with a simple file system (e.g. network share on a LAN or private WAN). |
| New | Json Index File Export Options | All Subjects in specified watchlists are to be synchronized to Json index on the file system. Specified watchlists only  Need to be able to configure to a fine level, what metadata is exported, e.g. should be able to export subject firstname/lastname, or just the photo.  This config COULD (not must) be flexible – e.g. if the user wants to use more custom fields, they could choose additional fields from NFW, following the Json data structure from Subject to do this dynamically. |
| New | Watchlist Sync | Sync. Users should be able to define whether sync is automatic or manual (see below) |
| New | Sync Methods | Automatic Sync  Periodically (configurable, e.g. nightly) check for local changes to the shared folder. Commit and push them to Azure Blob Storage  Should be able to disable this automatic sync. |
| New | Sync Methods | Manual Sync  User is able to manually force immediate sync of files to Azure Blob Storage |
| New | GUI | New GUI page in Nexus to manage watchlist syncing, including:   * Configuration * Monitoring (could this go into NFW System monitor?) * Reporting – synchronisations per day, MB synced up/down etc (could this go into NFW report module) * Sync Configuration (e.g. interval configuration and manual sync), Time Server to coordinate with, etc. |
| New | License | New License restriction in Nexus to restrict usage of watchlist syncing  e.g. restrict based on total number of subjects you can sync, and time.  e.g. restrict based on role: Contributor vs Recipient.  Contributors would also need to pull – so you could have say, 3 Contributors and 50 Recipients. |
| New | Race Condition Handling | The design principal is one or few writers, many readers. From this standpoint, it is acceptable for the index file to be exclusively locked for writing while an update is made, and then released, allowing one writer to make changes at any one time. |

### Contributor Processes - Seeding

Given: Blob store is empty, no index file exists

When: user configures Nexus 2.0 for the first time and attempts a sync, an error will occur that there is no index file available.

Then: user is given an option in the Nexus 2.0 GUI to either correct their configuration, or to “seed” the initial blob storage data using a specified watchlist with existing subjects.

### Contributor Processes – Re-Index

Per interval (e.g. once a week), the whole system index should be checked. This means local DB subjects vs index subjects, correcting any differences.

Re-index should be available on demand for human operators.

### Contributor Process – Startup

On startup Nexus 2.0 should look for “New Subjects” (see below definition), and add them to the index if they are not already added. Otherwise the usual index checks should be made.

Full re-index should not be the default behaviour on startup.

### Contributor Processes - Updates and Conflict Resolution

A screenshot of a computer

Description automatically generated

Figure 4 – Contributor CRUD

When: User makes a change to a subject in a synced watchlist

Then: change is detected by Nexus 2.0, Nexus 2.0 attempts to push the changes to the central Repository as individual differentials.

Alt-When: Should any differentials fail to be pushed for any reason, Nexus 2.0 will maintain a record of the change locally in a ‘Local Differential Log’ until sync time arises, at which point it will attempt to sync them once again.

When: Multiple changes happen to the same subject (e.g. create, update, update), before the contributor push or sync process has triggered

Then: (SHOULD) the latest change is treated by Nexus 2.0 as a ‘merged change’. This means that instead of three differentials pushed to the central repository for the same Subject, only one differential is pushed – the merged change. This change merging only happens when the differential fails to be pushed to the central repository immediately.

|  |
| --- |
| **Note: What is a ‘Differential’?**  An individual change JSON file with the updated Subject information presented in the same JSON structure as the Index JSON. One  **What is the ‘Differential Log’?**  The ‘differential log’ is a temporary a file structure of individual JSON files, one for each subject, and associated images. The Json files include the same metadata as an individual Subject in the Index Json.  The Central repository has the primary differential log that all contributors push changes to. |

When: **push time** arises OR immediately when change is detected

Then: contributor uploads ALL the deltas to the /Deltas/ folder in the storage account (or file system)

And Then: contributor deletes its local delta files ONLY when successful write of the Delta has been assured (Assured delivery).

When: A Contributor triggers a sync process (configurable schedule)

If: Contributor finds Deltas

Then: contributor attempts to acquire lock on index file

[See reference for how to do this: [Lease Blob (REST API) - Azure Storage | Microsoft Learn](https://learn.microsoft.com/en-us/rest/api/storageservices/lease-blob?tabs=azure-ad)]

When: contributor successfully acquires lock on index file

Then: contributor pulls down the delta changes from the remote and then applies any of its un-pushed delta changes

If: any single record has been changed remotely AND locally, then the most recent change “wins”, determined by timestamp of the Delta file

Alt-When: a Delta with a DELETE is applied before an UPDATE delta, the update DELTA will be ignored as the DELETE takes priority.

**Alt-When**: contributor is unable to acquire lock on index file to synchronise the index

Then: Synchronisation will be skipped until the next Synchronisation time arises, and a WARNING will be recorded in the technical log files.

### Index Update Process by Contributors with “Index Maintainer” Role

A diagram of a flowchart

Description automatically generated

Figure 5 - Index Maintenance Process

Not all Contributors are equal.

When: a contributor also has the role of “Index Maintainer” (restricted by license)

Then: a contributor has the responsibility of processing Deltas, if any are discovered in the /Deltas/ directory of the file system/Blob store

Index Maintainers should have a more frequent checking interval and are more trusted

Index Maintainers report warning and errors to users – e.g. if a delta was retrieved but it was over 30mins old (say), then it may raise a warning for users to check in with the remote contributor site.

### Subject Deletion Process

When: Deletion is detected by Nexus in a contributor system.

Then: Deletion deltas are uploaded straight away

When: A Contributor triggers a sync process (configurable schedule)

If: Contributor finds Deltas

Then: contributor attempts to acquire lock on index file

Then: Contributor processes the deletion deltas and applies them to the deletion index file [See reference for how to do this: [Lease Blob (REST API) - Azure Storage | Microsoft Learn](https://learn.microsoft.com/en-us/rest/api/storageservices/lease-blob?tabs=azure-ad)]

When: contributor successfully acquires lock on index file

Then: contributor pulls down the delta changes from the remote and then applies any of its un-pushed delta changes

**Alt-When**: contributor is unable to acquire lock on index file to synchronise the index

Then: Synchronisation will be skipped until the next Synchronisation time arises, and a WARNING will be recorded in the technical log files.

### Deletion Index File Maintenance Process

When: Year changes, e.g. from 2023 to 2024

AND WHEN: Next deletion workflow process starts

Then: The contributor creates a new Deletion Index file for the current year, and records the updates in there.

When: other contributors make their updates in the new year 2024

Then: all further deletions are recorded in the new year’s deletion index

### Detecting Deleted Subjects

Deleted subjects are either

* those whose GUIDs are listed in the Deleted Index JSON
* OR those who are detected as deleted from the local NFW Database: MS SQL “Change Data Capture” feature <https://www.c-sharpcorner.com/UploadFile/db2972/change-data-capture-in-sql-server/>

### Detecting new Subjects

New subjects are those that either

* Appear in the Index Json AND are not in the deleted subjects index AND are not yet present locally; or
* OR those who are detected as newly input from the local NFW Database: MS SQL “Change Data Capture” feature <https://www.c-sharpcorner.com/UploadFile/db2972/change-data-capture-in-sql-server/>

### Detecting Updated Subjects

Updated Subjects are those that either

* Appear in the index JSON and are also present in the local NFW system, but whose data has been modified (determined by an MD5 Hash of the subject). If this MD5 hash differs, the local subject is updated to match the remote
* are detected as updated from the local NFW database: MS SQL “Change Data Capture” feature <https://www.c-sharpcorner.com/UploadFile/db2972/change-data-capture-in-sql-server/>

### Contributor Process – One-Push Delete

I must be able to push a button to wipe all subjects in the central Azure storage, and clear the index. Process should ask me to confirm by showing a pop-up and then having me type the word “DELETE” into the box.

### Lock/Lease Process

Given: contributor system attempts to acquire a lease/lock of the blob

When: the lease is successfully secured

Then: The lease is later freed/released by the contributor programmatically releasing the lease

OR Then: Should the lease not be returned by the contributor (e.g. because of some local problem), then the lease is automatically released by Azure Blob Storage. [Lease Blob (REST API) - Azure Storage | Microsoft Learn](https://learn.microsoft.com/en-us/rest/api/storageservices/lease-blob?tabs=azure-ad)

### Exception Handling – Disconnection

When: in an exceptional situation where the local contributor system is disconnected from the central repository, and for some reason the local site has continued to make updates locally while at the same time the remote has been updated.

Then: When the local contributor system is reconnected and finds conflicts between its LDL and the central index

Then: Nexus 2 doesn’t automatically overwrite the index, but instead marks them for human review.

## Contributor GUIS

### Contributor GUI – View Synced Subjects

I should be able to see the full list of synced subjects at the click of a button (like running a report)

I should be able to query the Azure blob store to see any files in the remote that are out of sync with the local system. (e.g. remote is newer than local, or local is newer than remote). From here I should have a button to either a) update local system to remote file, or b) push local file to overwrite remote)

I should be able to manually obtain or release a lock on the index file in the central repo

I should be able to enter “maintenance mode” to disable live-updates of the watchlist from my system.

I should be able to query the Azure blob store to see any files that are not currently tracked by the index.

I should be able to one-click delete one or more, or all files that are not tracked by the index

### Contributor GUI – View and Resolve Sync Errors

I should have a webpage GUI where I can see a list of all the subjects that are in conflict and require human review.

I should be able to select multiple subjects on this screen to ‘mark reviewed’ and push.

From this screen I should be able to open a specific subject to view its differences (see below)

### Contributor GUI - View Sync Differences

This view is accessed from the list of sync errors (above)

I should be able to see a merged view of the local vs remote subject metadata. Local subject records on the left, remote on the right.

When this screen loads, the remote subject must be the latest version of the remote subject, including any deltas.

|  |  |
| --- | --- |
| Local Subject | Remote Subject |
| Photo(s) | Photo(s) |
| Firstname | Firstname |
| Lastname | Lastname |
|  |  |

Here, I should have the option of selecting which records to “keep” as “merged”, and “Publish”. Publishing will immediately push a delta.

From this screen, I could be able to navigate forwards and backwards to the next and previous records for merging.

# Recipient-Specific Requirements

| **Recipient NFW System**  New or Existing Feature? | | |
| --- | --- | --- |
| New | Watchlist Sync | Service regularly (configurable interval) checks for changes to data in the central Azure Blob Store. For each changed file we update the local NFW system, whether that is to create, read, update or delete etc the local Subjects in NFW  Even small changes are detected and synced, such as an individual image, or even data such as last name. |
| New |  |  |
| New | GUI | New GUI page in Nexus to manage watchlist syncing, including:   * Configuration * Connect to Azure Blob Storage * Role (Contributor vs Recipient) (restricted by license) * Which watchlist to pull into (if different from the remote) * What local directory to use (for the locally managed files / temp staging area) * Monitoring (could this go into NFW System monitor?)   Reporting – synchronisations per day, MB synced up/down etc (could this go into NFW report module) |
|  |  |  |

### Handle Difficult Face Images

With the Index file are included face metadata such as head position x,y,w,h and eye position (left/right).

Nexus could use this information to detect faces that it otherwise fails to detect, e.g. because the resolution is too large/too small, or other faces are detected in the image.

### Readonly Watchlists

Update NFW-specific to make a specific watchlist readonly from the UI, apart from limited users.

# Shared GUI Requirements

Whether recipient or contributor, I should be able to

### Installer – [Later]

Can be script install for now

### Re-Index on demand

I should be able to trigger full re-indexes when I want to, at the click of a button.

### Update Index on Demand

I should be able to manually trigger a differential update of my system when I want to, at the click of a button.

### View System Status

The system can have different states.

* Running or stopped (local system service)
* Pending Diffs (on remote) (I should have a button to “peek for changes”, which will give me a count of the diffs)
* Synchronising Diffs (live)
* Doing a full re-index

#### Contributor-Specific System Status View

I should be able to see a list of any pending deltas to submit (in local log) – along with any errors or warnings about these.

### Configure License

This should be to Nalpeiron license manager? Or LMX

### Configure Central Repository

Different types of central DB backend should be supported. Out of the box it should be Azure Blob Storage, or File System.

I could have 1 or N central watchlists synchronised.

Full Rebuild index schedule

Refresh index schedule

# General Technical Requirements

### Authentication

I must support

* Certificate Authentication ([Microsoft identity platform and OAuth 2.0 authorization code flow | Microsoft Learn](https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow#request-an-access-token-with-a-certificate-credential) and [Client assertions (MSAL.NET) - Microsoft Authentication Library for .NET | Microsoft Learn](https://learn.microsoft.com/en-us/entra/msal/dotnet/acquiring-tokens/web-apps-apis/confidential-client-assertions))
* Client secret Authentication

### Auditing – I must have the “reason for processing” captured

we already do this with NFW Audit logs

### Auditing

1. Every user action must be audited locally
2. Every workflow action must be audited locally (e.g. pushes, reindexes etc)
3. Every DELTA must be saved/kept on Azure for at least 90 days (Azure config)
4. A 90-day history of Blob Versions could be maintained (Azure config)
5. Azure Application Insights must be configured for API Management to enable auditing of all requests to the central DB store. This is described/proven in this document, section “*12 - Auditing Client Activity hitting the Central Repository*” (click)

### Development Approach – Plugin Architecture

In Nexus 1.0 we “patched” NFW. Instead we should make NFW “plugin friendly” so that we can add/remove plugins ike Nexus 1.0- or Nexus 2.0 as needed, without overiwriting pages in the Dashboard.

Nexus 2.0 should appear as a new button in the navigation menu.

Nexus 2.0 services should appear in NFW Services Monitor

# Key Decision Log

|  |  |
| --- | --- |
| Date | Decision |
| 2023-10-11 | All Photo data will be stored as JPG |
| 2023-10-11 | For a contributor making updates, the DateTime for an update will be determined by the file age according to the file server.  In this way, if there is any field conflict then the “last update wins”. Even if the changes occurred in a different order in reality at the remote sites. The ground truth has to be the central database.  For these reasons, a contributor MUST regularly synchronise its local updates. |
|  |  |
|  |  |

# Notes from Internal meeting 2023-07-14 Craig, Dhananjay, Luke, Mridul

* Legality of sharing data is responsibility of the customer that is administrating the system. This expectation has been established with the end customer (they already have this in place)
* Term “Central Authority” is a group of 1 or more contributors (say, 3), that are responsible for the data in the central watchlist.
* Central Data Storage technology
  + Blob Storage is proposed but Nexus 2.0 is intended to be adaptable so we can still change this for a better solution if we can settle on one.
  + We don’t anticipate making money from central system, so needs to be simple and robust.
  + Must have: easy GUI for central authority to add/remove/update contributors and readers. This GUI should come with the central storage technology, we don’t want to build this.
  + Should have: user-friendly GUI where I can see all subject files, their current version, their current data, and any previous versions. This GUI should come with the central storage technology, we don’t want to build this.
  + Must have: an open standard API that anyone can understand and easily integrate, e.g. REST. Again, we don’t want to build this.
  + Must have: an open standard, industry trusted data format. NIST is proposed, but JSON could be more appropriate.
* Central Storage Workflow considerations:
  + Recipients only ever need to take latest version of any given Subject.
  + Contributor conflict resolution is – “latest version wins”.
  + Storage needs to keep version history of files for a short period of time (e.g. so support can see what happened if two contributors did write at the same time)
  + Contributors will be responsible for auditing contributions.
  + Contributors responsible for the quality of the data going into the central system.
  + Recipients will be responsible for auditing “reads”
  + Nice to have: quota read/write limits enforced by the central storage.
  + Nice to have: audit logs at the central storage, of who did what and when.
  + Nice to have: option for read costs to be borne by recipient system only (useful for future sales but not for Australia opportunity)
* Regarding image processing issues (e.g. not all FR systems will be able to enrol a particular image)
  + Customer accepts that this individual might not be in all biometric systems across the estate
  + Customer attempts to troubleshoot issues on a case-by-case basis to improve the service
* Error Scenario: Recipient fails to process a new file (e.g. face not detected, too many faces detected, bad file, connection to storage lost, etc)
  + Log error to technical log
  + Save the failed file to a staging area
  + Local system will not have the update
  + Human resolution:
    - Technical teams detect the error during ITHC (or other error reporting dashboards out of our scope)
    - Technical teams attempt to resolve the error themselves
    - If they can’t, technical team raises a support ticket with the central authority service desk
    - Central authority service desk takes resolution action
    - Customer accepts that this individual may not be registered correctly in all systems

# Auditing Client Activity hitting the Central Repository

All activity by third party clients should be audited by Azure. This ensures accountability that can be audited by the organisation managing the central Azure Repo.

To do this, Application Insights must be enabled for the Azure API Management (out of scope here but CST know how to do it)

One thing we do need to check is that the Api Management is set to sample 100% of all requests and that headers are also logged (to record filenames)

Headers that should be logged include

* Blob
* Container
* LeaseID

A screenshot of a computer

Description automatically generated

Then, when you open up Application Insights you can run queries which can be manually reviewed and mined for insights and audit purposes.

This allows the central authority that is managing the system to:

1. Manually validate that all clients are correctly checking in to get index data
2. Check the activity for a specific client
3. Check the activity for a specific subject “have all my clients taken new subject with ID xyz?”
4. Check whether a specific client has downloaded a specific subject (and when)

## View all requests

Run the following query:

|  |
| --- |
| requests | where name has "put " and resultCode ==201 |

Allows us to see all the requests and even what files were worked on (GUID indicates file). Below I can see that luke-alienware uploaded a delta

A screenshot of a computer

Description automatically generated

…and later downloaded

A screenshot of a computer

Description automatically generated

## View all requests from a particular client

Filter your query to get all requests from a particular client:

|  |
| --- |
| requests | where customDimensions has "Luke-PC" |

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Note: Subscription Name is unique Identifier for a client

Note – the Unique identifier is the subscription name as you can see below. This means it is important that this is kept similar to the Display name for the subscription, to help us find it easily from the list here, e.g. using CTRL-F to ‘find in page’

A screenshot of a computer screen

Description automatically generated