IDV Feature - Validation of Data with Authoritative Source

Jira Issue(s)	IDV-474: IDV - Validation with mock authoritative sources (Backend) DONE IDV-518: IDV - Validation broker to authoritative sources (Backend) DONE IDV-1426: IDV - Validation of Form Data Evidence (Frontend) DONE
Target Delivery	20 Dec 2023 deferred to 30 Apr 2024
Product Owner	@Patricia Wiebe
Team	@Alec Laws @Shane Fright @Lauren Dewar (Unlicensed) @Manpreet Sandhu @James Carter @jona.kadhe (Unlicensed) @Opeyemi Idris
Summary	This feature will define the ability to validate form data with an external service, to facilitate validating user-entered data with an authoritative source in a verification process.
Status	Alpha deferred to Beta

Table of Contents

Problem Definition

Objective/Outcomes

We will provide a way to integrate with a backend validation service to confirm a person's data exists and matches in an authoritative data source (registry). The specific external validation service will be configurable per customer/client. Each customer will likely provide and use a different validation service, though it's possible that a validation service is used across multiple

clients of the same customer, and possibly across multiple customers with appropriate agreements in place.

Our approach is to make a broker and interface to each customer's backend validation service. This broker and interface will be called from the IDV service based on configuration if/when form data is collected in a person's verification flow, or possibly during an agent's workflow.

We will also create and use mock services in place of real customer's backend validation services.

Success metric

This feature will be successful if:

- an end-user's form data can be validated with an external backend validation service, as part of the verification process
- the IDV service is able to interface in a dynamic way with a variety of external validation services
- the IDV service can be configured with which external validation services will be called in a verification plan
- the IDV administrator can register new external validation services on behalf of a client to use in their verification plans without needing re-deployment of the Verification API (assumption: they also deploy a new microservice broker)
- invalid form data that does not meet requirements is rejected by the Verification API (internal validation before external validation)
- invalid form data that does not pass the external validation service is not stored and an error message returned to the frontend application

Scope & Constraints

In Scope:

Frontend Changes

- Display error messages provided from backend validation services
- Upon error, allow end-user to change data fields and try (upload) again
- Include context and label on all multipart form data file uploads

Mock Backend Validation Services

- Creation of two example (mock) data validation services that could represent our customer's authoritative sources
  - PEI Mock DL/VID search API

    - PEI Mock HC validation

Backend Changes

- An internal service within the Verification API that parses XState configuration related to validation of captured form data (internal validator)
    - ➡️ ⚠️ call internal validator when form-capture data is uploaded
- Microservice validators that broker the interface to a (mock) external validation service, and present a standard interface to the verification API (external validator)
    - ➡️ ⚠️ call external validator(s) when form-capture data is upload and the internal validator has passed and the validators have been configured for this form-capture data
    - ➡️ ⚠️ call external validator(s) after a reviewer corrects data from this form-capture data
- Require extra params on file upload (context, label) to determine if validation services should be run
- Data model change to add which validation services exist
- Data model changes to add a relationship between clients and the data validation services they can use (external validators)
- Data model changes for the internal validation service (form-capture internal validation)
- Way to pass external validation service error messages back to the frontend React application for display to an end-user

Out of Scope:

- Any other changes to the reviewer portal application
- Any real integration work with customer APIs
- Any internal validation service other than the plan-based form-capture one
- Rate limiting within the Verification API or validation broker before sending to external data validation services
- Validation or other types of processing of captured photos or videos with other backend services
- Validation with form data collected from multiple frontend steps (could be future PRD)

Constraints:

- Implementation should align with the general state architecture defined in IDV Verification Flow using XState
- Is dependent on the completion of IDV Flow - form-capture State (Enter data from ID Card)

Solution Definition

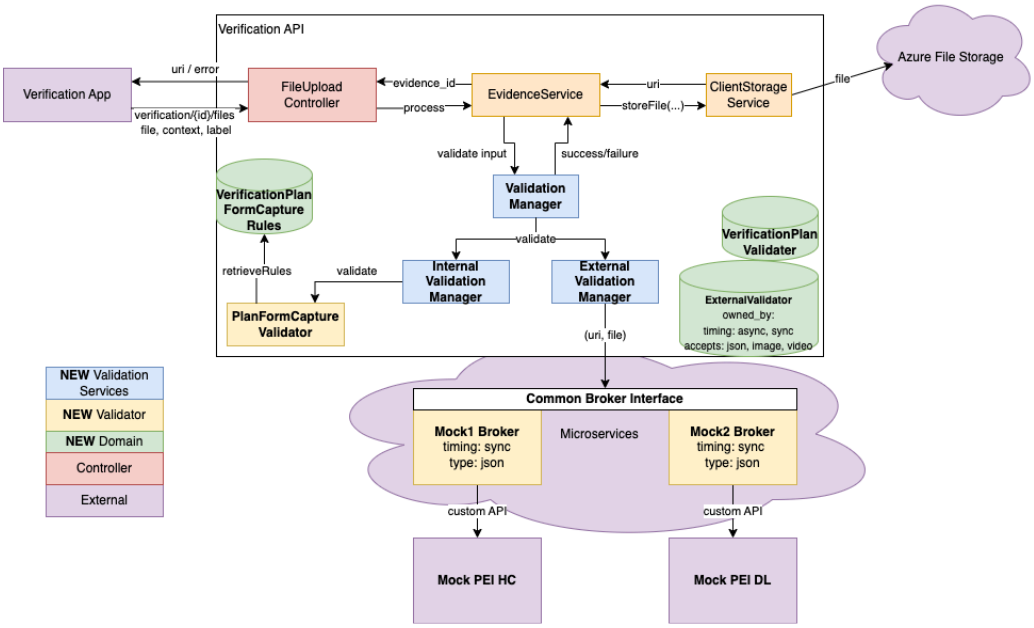
Terminology

Term	Definition
Internal Validation Service, aka internal validator	An internal service to the Verification API that can be used to validate a set of data We will create one - to parse the XState configuration (verification plan) for form-capture rules (max length, required, regex) and to apply those rules to the uploaded form data. This will be invoked immediately after form-capture upload, and before sending to an external validator.
External Validation Service aka external validator	An external service to the Verification API that can be used to validate that a set of data exists, such as a health card or a drivers licence. Examples include: • DIGWY used in the OTA project to look up health cards • PEI Client Registry to look up health cards • PEI SWAT to look up drivers licence / voluntary ID cards.
External Validation Service Broker aka broker	An externally hosted service that acts as an intermediary between the Verification API and specific external validation services See diagram below - Mock1 Broker

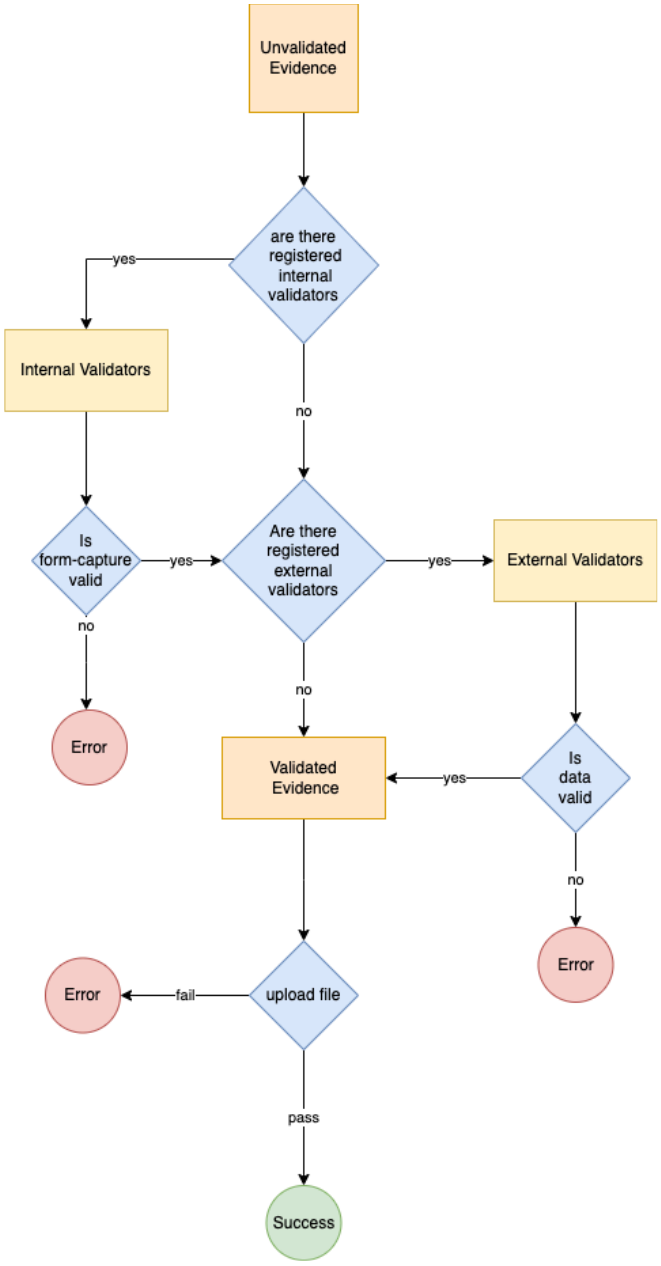
Architecture

The following diagram illustrates how form data is processed through the Verification API to be validated. If the data validation fails, an error message is displayed to the end-user in the frontend application and the end-user can try again.

Architecture:



Logic:



Defining External Validation Services in XState

In our XState model, data validation could be an extension of any state type. We will extend the metadata in the `form-capture` state within to specify which external validation service(s) to use in a verification plan.

Validation

An array of names for external validation services to be triggered after uploading the form data. If the validation metadata is not provided or is empty, there are no validations to process.

- ➡️ ⚠️ invoke when form-capture data has been uploaded (and after internal validator has passed)
- ➡️ ⚠️ invoke after reviewer corrections to this form data

```
"state name": {
 meta: {
   "type": "form-capture",
   "i18n": { ... },  // dynamic page content
   "fields": [ {} ], // form field definitions
   "validators": [ "<external validator service names>" ]
 },
 on: { ... }   // name of next state
}
```

For example, we might have form data pass through two validators, specified consecutively in an array.

```
"validators":  ["service-id-1", "service-id-2"]
```

Backend processing of an XState Verification Plan

XState / Front end concept: State.{identifier} = backend context

1. build list of allowed context + type pairs for valid file upload and create `ApplicantPlanAllowedContext`
   a. get each state identifiers (“ `instructions` ", `"id-card"`) and use as specific context
      i. within states, use meta.type as the type
2. link validation services to pairs
   a. Internal validation
      i. look for form-capture types
      ii. for each, transform `meta.fields.rules` into `ApplicantPlanFormCaptureRule` domain
      iii. return error if unable to transform (rule type not found) OR skip
   b. External validation
      i. for every type, look for `validators` element
      ii. for each, transform array into `ApplicantPlanValidator` rows
         1. if `service-id-1` is not found in `ExternalValidator` domain, return error
         2. if client does not have access to specified `ExternalValidator`, return error

Validation Broker Services

Each external validation service will have a microservice broker that exists as a **separate** service from the Verification API Application. It is deployed separately and security can be internal, API key based, or both.

*no ingress and has to use internal DNS (internal URI)

*API key (internal or external URI)

Each broker will be registered with a unique service identifier as an `ExternalValidator`. It will implement a common API (multipart file) to the Verification API. The service identifier can be specified in the XState configuration. The Verification API will pass data files to a broker, then the broker will determine how to implement the specific validation using data within the file, passing that data to their validation API.

Error cases:

If the internal or external data validation service fails the validation request, the validation broker will respond with the same 4xx status code and error to the front end:

```
HTTP/1.1 4xx

{
  "error": "validation_error"
  "message": "Cannot handle .png file type"
}
```

- Any 401, 403, 404, 405, 407, 410 (rate throttling) status codes from an external broker will be transformed by the Verification API into a `500 Internal Server Error`.

➡️ ⚠️ - Other 4xx status codes and description will be propagated without change to the react app

- 5xx status codes will be transformed into a `500 Internal Server Error` error.

The exact list of 400 level errors is left out of scope as they are defined by the validation API, the front end is expected to handle any 400 level error gracefully.
Errors must be logged appropriately including which broker failed and which verification_id was attempted. (no PII)

External validation broker micro-services which receive errors from the downstream API should log appropriately how it failed. (no PII)

➡️ ⚠️ Messages from the external validation broker will be provided to the FE, such that the message could be displayed to the user. The “message” should be delivered as a “message

code" that the FE can translate into a user-friendly message, with language localization. Messages to the FE could be errors about system availability or that the user has exceeded the number of attempts or that they simply did not pass validation.

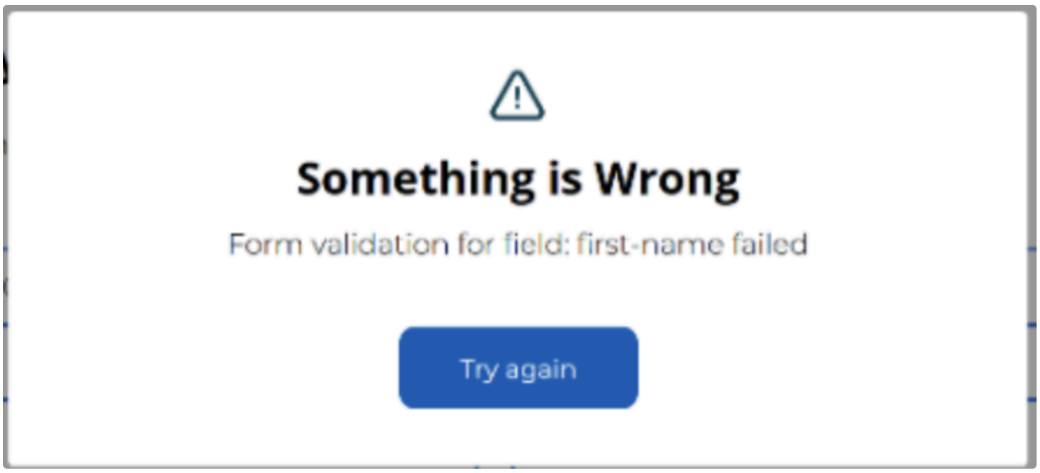
```
HTTP/1.1 502

{
  "error": "validation_error"
  "message": "validation service is unavailable or misconfigured"
}
```

what has been implemented - noted July 31, 2024:

```
{
    "error": "validation_error",
    "errorDescription": "Another account with this card already exists.",
    "lookupCode": "010.007.000",
    "status": 409,
    "timestamp": "Tue Jul 30 13:03:46 EDT 2024"
}
```

and the `errorDescription` field shows up in the front end like this:



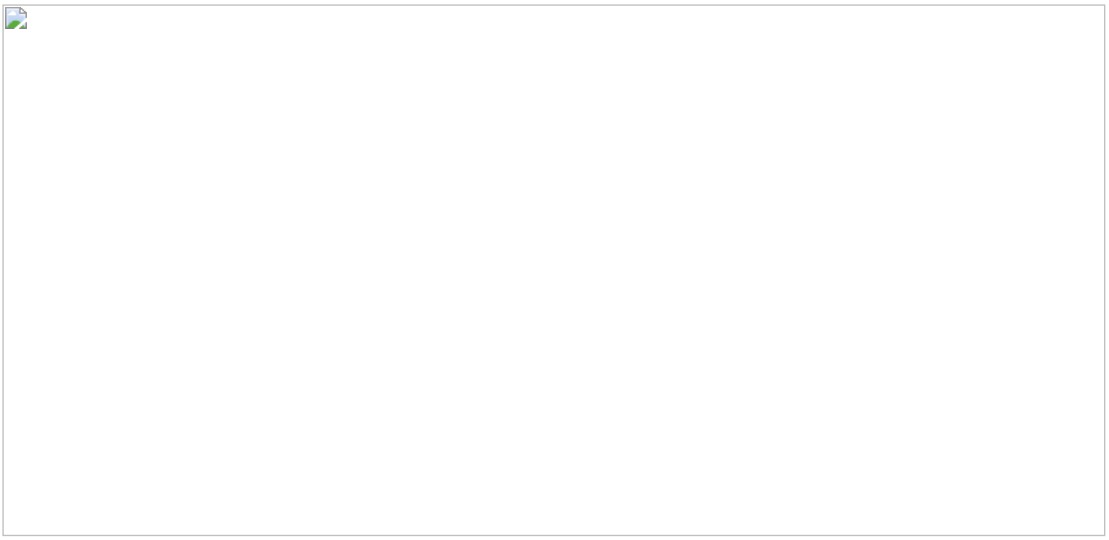
Responsibilities of the Verification API Application:

- Confirm that form data was received
- Using internal validator to confirm form data meets requirements per verification plan
- Calling appropriate brokers for each validation requested, per XState configuration
  - Providing form data as a JSON data file to the broker
- Return an appropriate error message if validation(s) do not pass
- Store form data if validation(s) pass

Responsibilities of a External Validation Broker:

- Receive requests for validation through multipart form data
- Call validation functions using external validation service API(s) and/or internal mechanisms
- Converting JSON data file into format expected by the external data validation API
- Interpreting the response from the external data validation API
- If validation fails, transform to an error message that can be returned for display to end user
- Transforms all 404 not found error to a 200 error with error message So the IDV api can translate it properly to error message the Frontend understands.
- Any additional internal validation checks such as minimum age requirements

Sequence Diagram



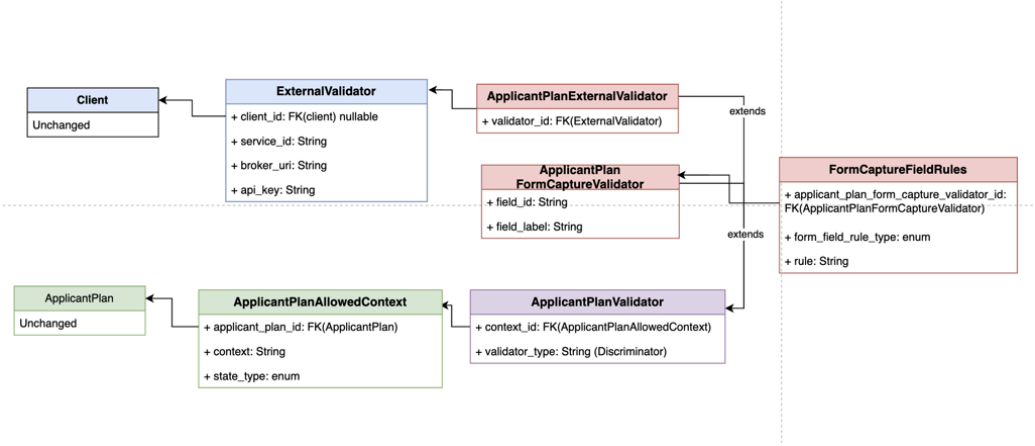
Data Model Changes:

New:

- ExternalValidator
- ApplicantPlanAllowedContext
- FormCaptureFieldRules
- ApplicantPlanValidator(base table)

  extended by:

  ApplicantPlanFormCaptureRule
  ApplicantPlanExternalValidator



Decision Log

	Question	Decision made	Decider	Date
1	Do we externalize the validation APIs or build them into the Verification API (see alternate architecture above)			
2	Do we require client plans to specify which API they want to use, or do we limit clients to have a m1-1 relationship between document (form data set) + DVAPI (as in any one client would only only be able to use one specific DVAPI for any one card).			
3	Does the backend need to validate the form data, and if so can it do so with a standard style validation service?	• 1. It will be needed by clients who wish to sanitize evidence they'll accept, but left OOS and validation is done at customers microservices as needed (Shane concern: clients should be confident they'll always get back information that matches their requirements in a plan)		

		2. Implement a basic internal validation service for all clients that relies on the xstate plan.field.rules section, (James: worry on regex validation performance) 3. Implement #2 as a microservice that takes in a set of rules + form data (same concern for regex)		

User Stories

User Stories

Persona + **Need** + **Purpose** = User Story

Acceptance Criteria is what ‘done’ means

	User Story	Acceptance Criteria
1	IDV-474: IDV - Validation with mock authoritative sources (Backend) DONE	
2	IDV-490: create Mock PEI validation service, like PEI staging validation service DONE	Acceptance criteria • takes same inputs as PEI staging validation service • gives same outputs as PEI staging validation service • provide a way to generate both pass and fail results
3	IDV-518: IDV - Validation broker to authoritative sources (Backend) DONE	
4	IDV-519: As a person presenting form data for my identity verification, I can see an erro	• This should not happen if the frontend application is ensuring the form data meets requirements

	r if my form data does not meet the requirements of the verification plan DONE	before uploading; so this is really a check if the frontend application is doing what it should • ➡️ ⚠️ ensure this only applies to form-capture state
5	IDV-520: As a malicious user using the Verification API instead of the React application, I can see an error if my form data does not meet the requirements of the verification plan DONE	Acceptance criteria • required field values exist • if field value exists, it does not exceed max-length • if field value exists, match regex
6	IDV-521: As an IDV administrator, I can configure a new validation broker in the IDV database, so that it can be referenced in a verification plan DONE	Configure the mock PEI validation service Acceptance criteria • able to configure and use it (within a client's verification plan) without needing to redeploy the Verification API component
7	IDV-522: As an IDV administrator, I can configure how to connect to the validation broker through the IDV database DONE	
8	IDV-523: As a person presenting form data for my identity verification, I can see an error if my form data does not pass validation with the configured external validation service DONE	Acceptance criteria • get error if form data does not match, e.g. record not found • error message comes from the validation broker, which may get error message from the external validation service itself • JSON data file not stored in client-specific storage account
9	IDV-524: As a person presenting form data for my identity verification, I will not see an error if my form data should not pass validati	Acceptance criteria • the user would not see any validation error message

	on but the external validation service is not configured DONE	
10	IDV-525: As a malicious user using the Verification API instead of the React application, I can see the same validation errors as if I used the React application DONE	Acceptance criteria • get error if form data does not match
11	IDV-526: As a person presenting form data for my identity verification, I can see an error if the validation broker is misconfigured for the external validation service (HTTP 403) DONE	Acceptance criteria • get error message
12		
13	epic - IDV - Validation error messages (frontend) IDV-1426: IDV - Validation of Form Data Evidence (Frontend) DONE	
14	As a person entering my data in a form, I want to be informed if my data is not validated/accepted so that I can try again	
15	IDV-1427: Front end is able to successfully upload a file through the backend when there is basic validation on form fields DONE	
16	IDV-1428: Front end is able to successfully upload a file through the backend when there is basic external validation through a broker DONE	
17	IDV-1429: front end is able to display error messages that correspond to different status codes returned by the backend when validations fails and allow user to change form field data DONE	

Launch Readiness

This feature will not be released or launched as a software product or service to customers at this time.

Impact

To be completed after work is done, feedback received.

Test Plan

(link to Jira test plan and Jira test set)