**OpenPlatform - Plan Documents Access**

**Feature Summary**

This feature will enable OpenPlatform users to access their plan documents. The project involves creating a new BFF (Backend-for-Frontend) for the OpenPlatform, which will integrate with the existing AWS-based Benefits Service currently used by Aetna Health. The solution will handle scenarios for displaying single, multiple, or zero available documents, **ensuring resource availability through feature flag checks, and integrating specific availability messaging in lieu of or in addition to documents.** It will also include support for language preferences.

The primary goal is to route the plan document list and PDF retrieval through a new dedicated OpenPlatform BFF to the existing AWS services. This approach benefits from existing capabilities like caching, which reduces the load on core services and provides a more resilient and faster user experience. The solution must also account for potential gaps in upstream data, such as missing plan names or document descriptions, and handle them gracefully. Search functionality is not in scope for the initial release but is on the roadmap, and the architecture should not degrade performance for future search implementations.

**Discovery**

Several options were considered for fetching the plan document list and retrieving the PDFs. The primary challenge is balancing app startup performance, on-page performance, and implementation complexity, especially considering the known data inconsistencies.

**Architectural Options Analysis**

Here's a breakdown of the considered architectural options, their descriptions, and a "meatball" chart summarizing their pros and cons.

**Option 1: Adapt Existing On-Demand Architecture (Chosen Direction)**

**Description**: This approach mirrors the existing Aetna Health architecture. A new route will be created in a dedicated OpenPlatform BFF. When a user navigates to the Plan Documents page, the client first performs **feature flag checks** to determine eligibility and message availability. If eligible for documents, the client calls the BFF, which in turn calls the backend Benefits Service to get the document list for that user using specific membershipResourceId and policyResourceId parameters. PDF retrieval follows the established pattern: the client requests a PDF via the BFF, which first checks an S3 cache and, if not present, retrieves it from the core endpoint and caches it for future requests.

**Option 2: App Startup Pre-fetch**

**Description**: This option proposes fetching the plan document list when the application starts up, based on the observation that plan metadata is sometimes missing. The data would be sourced from a plan doc index or API and cached on the client or in the BFF for the duration of the user's session. This would make the Plan Documents page load instantly.

**Option 3: BFF-Centric Caching & Augmentation (Hybrid)**

**Description**: In this model, the OpenPlatform BFF takes on more responsibility. It follows the on-demand pattern of Option 1 but introduces an additional layer of caching within the BFF itself, tailored to the OpenPlatform's traffic patterns. The BFF would be responsible for calling the backend Benefits Service and then augmenting the response with metadata sourced from another service or a static cache (e.g., "sourcing from app startup" data loaded into the BFF) before returning the final payload to the client.

**Meatball Chart: Pros and Cons of Architectural Options**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Option 1: Adapt Existing On-Demand Architecture** | **Option 2: App Startup Pre-fetch** | **Option 3: BFF-Centric Caching & Augmentation** |
| **Pros** | - Reuses proven architecture | - Extremely fast user experience on page load | - Fast app startup |
|  | - Minimal app startup impact | - Augments missing metadata centrally at startup | - Robust data handling/augmentation |
|  | - Leverages existing caching |  | - Optimized caching for OpenPlatform |
|  | - Lower complexity |  | - Simplified client-side logic |
| **Cons** | - On-page performance dependent on real-time API call | - Increases app startup time | - Increased BFF complexity |
|  | - Requires BFF/client to handle data gaps | - Data can become stale | - Potential for slight latency |
|  |  | - Increased complexity for app startup data fetching |  |

**Chosen Direction**

**Option 1** is the chosen direction due to its simplicity and direct reuse of existing, proven infrastructure. This aligns with best practices by prioritizing a phased approach, starting with a lean, functional solution that leverages existing, well-tested components. It reduces initial development time and risk by building upon a stable foundation.

To mitigate the risk of missing data, the OpenPlatform BFF will incorporate a **limited augmentation capability** (as described in Option 3) to normalize the data before it reaches the client. This hybrid approach ensures that critical display fields are present, without adding the complexity of full BFF-side caching in the initial phase. This approach aligns with the principle of "start small, iterate fast" while still addressing known data quality concerns.

**Solution Sketch**

The high-level flow involves the OpenPlatform client communicating with a new OpenPlatform BFF. This BFF will act as a proxy to the existing AWS-based backend infrastructure.

* **Pre-check with Features API:** Before attempting to retrieve any plan documents, the OpenPlatform client will first call a "Features API" to determine which types of plan documents are enabled for the specific membershipResourceId and to check for any special messaging (e.g., PlanDocsMsgAvail feature flag).
* **List Retrieval**: The client, having confirmed document availability through feature flags, constructs a request using the policyResourceId, membershipResourceId, and dateAsOf for the relevant plan type. This request is sent to a new endpoint on the OpenPlatform BFF. The BFF calls the existing Benefits Service /plan-document-list/retrieve endpoint, which checks its cache or calls core APIs to get the document list. The response is returned to the BFF, which may augment it and check for specific reasonCodes (like "HHL Not signed") before sending it to the client.
* **PDF Retrieval**: The client requests a specific document via the BFF, providing the documentId (which maps to communicationContentResourceId). The BFF calls the Benefits Service /plan-document/retrieve endpoint. This service first attempts to get the PDF from an S3 bucket. If it's a cache miss, the service retrieves the document from the core API, stores it in S3 for subsequent requests, and returns it.

**Orchestration Diagram (Conceptual)**

(For IA work/EDB Service Updates Only) This work involves creating a new BFF but leverages existing EDB service updates. The orchestration will be:

OpenPlatform -> OpenPlatform BFF OpenPlatform BFF -> AWS APIc -> Benefits Service Benefits Service -> Redis Cache (for list) Benefits Service -> S3 Bucket (for PDF) Benefits Service -> Core Plan Doc API (on cache miss)

**APIs/Swagger**

A new API contract will be defined for the OpenPlatform BFF based on the specific needs of the OpenPlatform, aligning with the data required from the existing native client APIs.

**New OpenPlatform BFF Endpoints**

* **Get Document List**: POST /v1/cvs/plan-documents/list
  + **Description**: Retrieves the list of plan documents for the user, based on provided membership and policy identifiers.
  + **Request Body**: The client will send membership and policy identifiers, along with a date. This mirrors the parameters typically used by native clients for similar backend calls.
  + {
  + "data": {
  + "policyResourceId": "3~141975+A+1",
  + "membershipResourceId": "5~235287654+10+1+20200601+141975+A+1",
  + "dateAsOf": "YYYY-MM-DD"
  + }
  + }

**Notes on Request Body:**

* + - policyResourceId: This should be from the same membership that is used to invoke plan summary. Should be either Medical or Dental membership as primaryPolicyType.
    - membershipResourceId: Use this value from the application startup API response for the selected membership.
    - dateAsOf: Formatted YYYY-MM-DD (no timestamp). For actively covered memberships, this should be <today’s date>. For pending coverage memberships (only supported for Medicare plan documents), this should be the coverageDatetimeBegin value from the application startup API response for the pending coverage.
  + **Response Body**: The response is a JSON object containing a data array of documents, tailored for the OpenPlatform, and may include a reasonCode for special cases.
    - **Zero Documents Example (including HHL Not Signed):**
    - {
    - "data":[],
    - "reasonCode": "HHL Not signed"
    - }
    - **Single Document Example:**
    - {
    - "data":[{
    - "planId" : "751133+BA+2",
    - "documentName":"Plan Document (Spanish)" ,
    - "documentId":"70023~13910995",
    - "documentUrl" :"/v1/cvs/plan-documents/retrieve",
    - "documentType": "SBC",
    - "description": "Summary of Benefits and Coverage (English)",
    - "title": "2024-IFP-767941\_48161MO0210015-00\_SBC.pdf",
    - "format": "PDF",
    - "documentSubType": "SBCIVLEX",
    - "language": "ENG",
    - "contentSize": "537838"
    - }]
    - }
    - **Multiple Documents Example:**
    - {
    - "data":[{
    - "planId" : "751133+BA+2",
    - "documentName":"Plan Document" ,
    - "documentId":"70023~13910995",
    - "documentUrl" :"/v1/cvs/plan-documents/retrieve",
    - "documentType": "Dental",
    - "description": "Benefit Plan Description",
    - "language": "ENG"
    - }, {
    - "planId" : "751133+BA+2",
    - "documentName":"Plan Document (Spanish)" ,
    - "documentId":"70023~13910996",
    - "documentUrl" :"/v1/cvs/plan-documents/retrieve",
    - "documentType": "Dental",
    - "description": "Benefit Overview (Spanish)",
    - "language": "SPA"
    - }]
    - }
* **Get Document PDF**: POST /v1/cvs/plan-documents/retrieve
  + **Description**: Retrieves a specific plan document PDF. All documents are expected to be PDFs.
  + **Request Body**:
  + {
  + "documentId": "70023~13910995"
  + }

**Notes on Request Body:**

* + - documentId: This corresponds to communicationContentIdentifier.resourceId from the Plan Document List API response for the document selected by the user.
  + **Response**: The raw PDF file (application/pdf).

**OpenAPI (YAML) Specification**

openapi: 3.0.3

info:

title: OpenPlatform - Plan Documents BFF

description: Provides plan document list and PDF retrieval for the OpenPlatform.

version: 1.0.0

paths:

/v1/cvs/plan-documents/list:

post:

summary: Retrieves the list of plan documents for a user.

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/PlanDocListRequest'

responses:

'200':

description: Successful retrieval of document list.

content:

application/json:

schema:

$ref: '#/components/schemas/PlanDocListResponse'

'400':

description: Bad request

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'500':

description: Internal Server Error

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

/v1/cvs/plan-documents/retrieve:

post:

summary: Retrieves a specific plan document PDF.

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/PlanDocRetrieveRequest'

responses:

'200':

description: Success - PDF file returned.

content:

application/pdf:

schema:

type: string

format: binary

'400':

description: Bad request

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

'500':

description: Internal Server Error

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

components:

schemas:

PlanDocListRequest:

type: object

properties:

data:

type: object

required:

- policyResourceId

- membershipResourceId

- dateAsOf

properties:

policyResourceId:

type: string

description: 'Policy Resource ID from application startup API'

example: '3~141975+A+1'

membershipResourceId:

type: string

description: 'Membership Resource ID from application startup API'

example: '5~235287654+10+1+20200601+141975+A+1'

dateAsOf:

type: string

format: date

description: 'Date for which the documents are requested (YYYY-MM-DD)'

example: '2025-06-28'

PlanDocListResponse:

type: object

properties:

data:

type: array

items:

$ref: '#/components/schemas/DocumentMetadata'

reasonCode:

type: string

description: 'Reason code for special scenarios (e.g., HHL Not signed)'

example: 'HHL Not signed'

DocumentMetadata:

type: object

properties:

webHyperlink:

type: string

description: 'URL for the communication content (from original backend)'

example: 'https://www.aetna.com/epublishing/getPlanDocument?serviceName=fetchPDF&nodeId=347812301'

documentType:

type: string

description: 'Type of the document (e.g., Dental, Medical, SBC, EOC)'

example: 'Dental'

description:

type: string

description: 'Description of the communication content, used for document title display'

example: 'Benefit Plan Description'

title:

type: string

description: 'Title of the communication content (e.g., filename from IFP)'

example: '2024-IFP-767941\_48161MO0210015-00\_SBC.pdf'

format:

type: string

description: 'Format of the document (e.g., PDF, HTML)'

example: 'PDF'

documentSubType:

type: string

description: 'Sub-type of the document (e.g., SBCIVLEX, EOCDTLS)'

example: 'SBCIVLEX'

language:

type: string

description: 'Language of the communication content (e.g., ENG, SPA)'

example: 'ENG'

contentSize:

type: string

description: 'Size of the content in bytes'

example: '537838'

communicationContentIdentifier:

type: array

items:

type: object

properties:

idSource: { type: string }

idValue: { type: string }

idType: { type: string }

resourceId: { type: string, description: 'Unique resource ID for the content' }

description: 'Identifiers for the communication content'

communicationContentEffectivePeriod:

type: object

properties:

dateTimeAsOf: { type: string, format: date }

dateTimeBegin: { type: string, format: date-time }

dateTimeEnd: { type: string, format: date-time }

description: 'Effective period of the communication content'

planId:

type: string

example: '751133+BA+2'

documentName:

type: string

example: 'Plan Document (Spanish)'

documentId:

type: string

example: '70023~13910995'

documentUrl:

type: string

description: 'Internal URL to retrieve the PDF via BFF'

example: '/v1/cvs/plan-documents/retrieve'

PlanDocRetrieveRequest:

type: object

required:

- documentId

properties:

documentId:

type: string

description: 'Corresponds to communicationContentResourceId in the backend (communicationContentIdentifier.resourceId)'

example: '70023~13910995'

Error:

type: object

properties:

httpCode:

type: string

example: '400'

httpMessage:

type: string

example: 'Bad Request'

moreInformation:

type: string

example: 'planId is a required field.'

**NFR (Non-Functional Requirements)**

* **Performance**: The solution must not degrade app performance. Caching at the Benefits Service (TTL 1 hour) and S3 layer will be leveraged to ensure fast responses.
* **Rate Limiting**: Existing rate limits for the backend services will be used. The Plan Doc List endpoint is 950/min, and the PDF endpoint is 2333/min. These will be monitored for the new OpenPlatform traffic.
* **Future-proofing**: The architecture must not prevent the future implementation of search functionality.
* **Scalability**: The chosen architecture (Option 1) reuses existing, scalable AWS services (API Gateway, Lambda, S3, Redis), ensuring that the solution can handle anticipated growth in OpenPlatform user traffic.
* **Observability**: Comprehensive logging, monitoring, and alerting will be implemented for the new OpenPlatform BFF, providing visibility into its performance, errors, and traffic patterns. This will include integration with existing APM (Application Performance Monitoring) tools.

**Services**

**Service Logic**

* **OpenPlatform BFF**:
  + Receives a request from the OpenPlatform client for the document list, including policyResourceId, membershipResourceId, and dateAsOf.
  + Calls the backend Benefits Service's /plan-document-list/retrieve endpoint, passing these identifiers.
  + Receives the document list from the backend.
  + Transforms the backend response into the format required by the OpenPlatform client, as defined in the API section. This includes:
    - Creating the appropriate JSON structure for the zero, single, or multiple document scenarios.
    - Appending the language (from the Benefits Service response) to the documentName for display purposes (e.g., "Plan Document (Spanish)").
    - Ensuring that if a description is not available, the field is not shown.
    - **Forwarding reasonCode**: If the Benefits Service response includes a reasonCode (e.g., "HHL Not signed"), this is passed directly to the client.
* **Benefits Service (Existing)**:
  + **List**: Logic remains as-is. It expects policyResourceId, membershipResourceId, and dateAsOf in its request. It checks a Redis cache based on a complex key (considering Medicare vs. Commercial, ANOC, etc., potentially including dateAsOf + policyResourceId + externalPlanId + classCode + priorAdjacentContractPBP for Medicare Group or dateAsOf + policyResourceId + priorAdjacentContractPBP for Medicare Individual, and dateAsOf + policyResourceId for IFP/Commercial) before calling the core API. The cache has a TTL of one hour. It returns reasonCode: "HHL Not signed" if applicable for self-insured plans without the signed letter.
  + **PDF**: Logic remains as-is. It expects communicationContentResourceId in its request. It checks an S3 bucket for the PDF. On a miss, it fetches from the core API, streams it to S3 for caching, and returns the document. It will also check the byte size of the file from core and return an error if it is 0 to avoid caching empty files.

**Server Stories (Backend)**

* **As an OpenPlatform BFF, I want to retrieve a list of plan documents for a given policyResourceId, membershipResourceId, and dateAsOf, so that I can provide it to the OpenPlatform client.**
  + **Acceptance Criteria:**
    - The BFF successfully calls the Benefits Service's /plan-document-list/retrieve endpoint with the correct policyResourceId, membershipResourceId, and dateAsOf.
    - The BFF handles various response scenarios from the Benefits Service (e.g., success, no documents, errors, including reasonCode like "HHL Not signed").
    - The BFF correctly transforms the Benefits Service response (including fields like webHyperlink, documentType, description, language, communicationContentIdentifier.resourceId) into the OpenPlatform API contract format (documentName with language, documentId, documentUrl).
    - The BFF gracefully handles cases where description or title are missing from the upstream response by not including them in the final documentName derivation or by hiding description field.
    - The BFF passes any reasonCode received from the Benefits Service directly to the client.
* **As an OpenPlatform BFF, I want to retrieve a specific plan document PDF by its documentId (mapping to communicationContentResourceId), so that I can deliver it to the OpenPlatform client.**
  + **Acceptance Criteria:**
    - The BFF successfully calls the Benefits Service's /plan-document/retrieve endpoint with the provided documentId.
    - The BFF correctly streams the PDF content received from the Benefits Service to the OpenPlatform client.
    - The BFF handles error responses (e.g., 400, 500) from the Benefits Service and propagates appropriate error codes to the client.
* **As the Benefits Service, I want to use Redis as a cache for plan document lists based on relevant identifiers (policyResourceId, dateAsOf, and specific Medicare attributes), so that I can reduce load on core APIs and improve response times.**
  + **Acceptance Criteria:**
    - The Benefits Service correctly derives and uses composite cache keys (e.g., dateAsOf + policyResourceId + externalPlanId + classCode + priorAdjacentContractPBP for Medicare Group, dateAsOf + policyResourceId + priorAdjacentContractPBP for Medicare Individual, and dateAsOf + policyResourceId for IFP/Commercial).
    - The Benefits Service checks Redis for existing cached planDocListResponse.
    - If a cache hit occurs, the cached response is returned.
    - If a cache miss occurs, the core API is called, and the successful response is cached in Redis with a 1-hour TTL before being returned.
* **As the Benefits Service, I want to use S3 as a cache for plan document PDFs, so that I can reduce load on core APIs and improve response times.**
  + **Acceptance Criteria:**
    - The Benefits Service checks the configured S3 bucket for the requested PDF using communicationContentResourceId as the key.
    - If the PDF exists in S3, it is streamed directly to the BFF.
    - If the PDF is not in S3, it is retrieved from the core API, stored in S3 for future requests, and then streamed to the BFF.
    - The Benefits Service verifies the byte size of the file from core; if 0 bytes, it returns an error and does not cache the empty file.
* **As the Benefits Service, I want to ensure my APIs are resilient and return appropriate errors and reasonCodes for various scenarios, particularly for HHL.**
  + **Acceptance Criteria:**
    - The /plan-document-list/retrieve and /plan-document/retrieve endpoints return standard HTTP error codes (e.g., 400, 403, 404, 500, 503, 504) with consistent error schemas.
    - For self-insured plans where the HHL is not signed, the /plan-document-list/retrieve API returns an empty communicationContent array with a reasonCode of "HHL Not signed".
    - The service correctly handles and propagates HHL Not Signed reason codes from upstream systems.

**Service Components**

* **New OpenPlatform BFF (Backend-for-Frontend)**: A new service specific to the OpenPlatform, responsible for handling client requests and communicating with the backend services.
* **Features API (Existing)**: An existing API consumed by the client to determine enabled features (e.g., DenPlanDocs, MedPlanDocs, VisionPlanDocs, PlanDocsMsgAvail) for a given membershipResourceId.
* **AWS Benefits Service (Existing)**: The existing microservice that contains the business logic for fetching plan documents.
* **AWS S3 (Existing)**: Used for caching PDF documents to improve performance and resilience.
* **Redis (Existing)**: Used by the Benefits Service for caching the plan document list (metadata).
* **Core APIs (Existing)**: The source of truth for plan documents when they are not in cache.

**Client Changes**

**Client Side Logic**

The client will integrate with the new OpenPlatform BFF endpoints.

* **Pre-check for Features and Messaging:**
  + When the user accesses the Plan Documents feature, the client will **first call a "Features API"** using the user's membershipResourceId (derived from application startup API data).
  + The client will check the Features API response for featureCode values such as DenPlanDocs, MedPlanDocs, VisionPlanDocs, and PlanDocsMsgAvail.
  + **Decision Point:**
    - **If PlanDocsMsgAvail is enabled OR if specific PlanDoc types (Medical, Dental, Vision) are NOT enabled for this membershipResourceId:** The client will **immediately display a "Special Messaging" UI** (e.g., an "HHL Not Signed" message or a "No documents available" message), bypassing the call to the BFF for document retrieval.
    - **Otherwise (if PlanDocsMsgAvail is not enabled AND relevant PlanDoc types ARE enabled):** The client proceeds to request the document list from the BFF.
* **Requesting Document List from BFF:**
  + The client will identify the appropriate policyResourceId and dateAsOf (today's date for actively covered, coverageDatetimeBegin for pending Medicare) for the selected membershipResourceId (based on the enabled DenPlanDocs, MedPlanDocs, or VisionPlanDocs features).
  + The client will call the new OpenPlatform BFF endpoint (POST /v1/cvs/plan-documents/list) with the policyResourceId, membershipResourceId, and dateAsOf in the request payload.
* **Processing BFF Response for Document List:**
  + Upon receiving the response from the BFF, the client will first check for a reasonCode (e.g., "HHL Not signed").
    - **If reasonCode: "HHL Not signed" is present:** The client will **display the "Special Messaging" UI** for the HHL scenario, even if some document data might have been theoretically returned (though typically data will be empty in this case).
    - **Otherwise:** The client processes the data array in the response:
      * If data.length is 0, the client must handle the "zero document" scenario gracefully.
      * If data.length is 1, the client should directly open the PDF view upon the user's initial tap.
      * If data.length is >1, the client will display a list of the available documents for the user to choose from.
* **PDF Viewing and Actions:**
  + The client will need to provide UI options for viewing, sharing, and printing the selected PDF document.
  + The UI/UX should follow the same design and behavior ("chrome") as other document views in the app.
  + The feature may need to be implemented as a full page rather than a sheet for better usability.

**UX Stories (Frontend)**

* **As an OpenPlatform user, when I navigate to the Plan Documents section, I want the system to first check if I'm eligible to view documents or if there's a special message for me, so I get the correct information immediately.**
  + **Acceptance Criteria:**
    - Client calls the Features API with membershipResourceId on page load.
    - If PlanDocsMsgAvail is enabled, or if no DenPlanDocs, MedPlanDocs, or VisionPlanDocs are enabled, the client directly displays a predefined message instead of attempting to fetch a list.
* **As an OpenPlatform user, if my plan requires a signed Hold Harmless Letter and it's not signed, I want to see a clear message explaining this, instead of an empty document list.**
  + **Acceptance Criteria:**
    - The client displays a user-friendly message for the "HHL Not signed" scenario.
    - This message is triggered either by PlanDocsMsgAvail feature flag or by the reasonCode: "HHL Not signed" from the BFF response.
    - The UI handles this gracefully, avoiding errors or blank screens.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there is only one plan document available (and no special messages), I want the PDF to open directly upon my initial tap, so I can quickly view it.**
  + **Acceptance Criteria:**
    - The client displays a "Plan Document" link/button.
    - Upon tapping, the PDF viewer opens immediately with the single document.
    - The PDF viewer chrome (design and behavior, e.g., share, print options) is consistent with other document views in the app.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there are multiple plan documents available (and no special messages), I want to see a list of documents with their names, so I can choose which one to view.**
  + **Acceptance Criteria:**
    - The client displays a list of documents.
    - Each item in the list shows the documentName (derived from description or title and possibly language).
    - If a description field is present in the API response, it is used for the document title; otherwise, it is handled gracefully (e.g., using title or a generic name).
    - Tapping on a document in the list opens its PDF in the viewer.
    - The PDF viewer offers options such as share and print.
    - All displayed documents are PDFs.
* **As an OpenPlatform user, when I navigate to the Plan Documents section and there are no documents available (and no special messages from PlanDocsMsgAvail), I want to see a clear message indicating that, so I understand the situation.**
  + **Acceptance Criteria:**
    - The client displays a user-friendly message indicating no plan documents are available.
    - The UI handles the "zero document" scenario gracefully, avoiding errors or blank screens.
* **As an OpenPlatform user, if I have documents in multiple languages, I want to see the language indicated (e.g., in parentheses) so I can easily identify the correct version.**
  + **Acceptance Criteria:**
    - The documentName displayed in the list includes the language in parentheses (e.g., "Plan Document (Spanish)").
    - The client includes attributes to allow for future filtering by language.
* **As an OpenPlatform user, I want the plan document viewing experience to be consistent in design and behavior, so I have a seamless user experience across the app.**
  + **Acceptance Criteria:**
    - The UI/UX for plan document access (e.g., buttons, layout, PDF viewer) adheres to the established OpenPlatform design guidelines ("chrome").
    - The feature may be implemented as a full page instead of a sheet for optimal usability.

**Field Mapping**

* **Client Request to BFF (POST /v1/cvs/plan-documents/list):**
  + policyResourceId: From application startup API response, corresponding to the selected plan's primary policy type (Medical, Dental, or Vision).
  + membershipResourceId: From application startup API response for the selected membership, associated with enabled feature codes.
  + dateAsOf: YYYY-MM-DD format. Today's date for Actively Covered. For Pending Coverage (Medicare only), use coverageDatetimeBegin from app startup.
* **BFF Response (PlanDocListResponse) to Client:**
  + data[].documentName: Derived by BFF. If description is available, use it. If language is available and not 'ENG', append (language) to the name. If description is missing, try to use title.
  + data[].documentId: Maps to communicationContentIdentifier.resourceId from Benefits Service response.
  + data[].documentUrl: Generated by BFF to point to its /v1/cvs/plan-documents/retrieve endpoint.
  + data[].documentType, data[].description, data[].title, data[].format, data[].language, data[].contentSize: Directly mapped from Benefits Service response if available.
  + data[].planId: Maps from policyResourceId related to the document.
  + reasonCode: Directly mapped from Benefits Service response if present.
* **Client Request to BFF for PDF (POST /v1/cvs/plan-documents/retrieve):**
  + documentId: Value obtained from data[].documentId in the Plan Doc List response.

**Any differences between platforms**

The logic and documentation must be consistent across Aetna and CVS implementations, and by extension, across Web, iOS, and Android platforms.

**Scenarios**

* **User navigates to Plan Docs:**
  + **Scenario 1: No PlanDoc features enabled for any membership:** Client displays "No documents available" message immediately (triggered by Features API).
  + **Scenario 2: PlanDocsMsgAvail feature is enabled (e.g., HHL not signed externally):** Client displays "Special Messaging" UI immediately (triggered by Features API), bypassing BFF call.
  + **Scenario 3: PlanDocsMsgAvail is NOT enabled, but HHL Not signed reason code is returned by Benefits Service:** Client displays "Special Messaging" UI for HHL (triggered by BFF response).
* **Single Document**: User has exactly one plan document available (and no special messages). Tapping the link opens the PDF directly.
* **Multiple Documents**: User has more than one document available (and no special messages). A list is displayed showing the title and description (if available).
  + **Sub-scenario: Multi-language documents:** List displays language in parentheses.
* **Zero Documents**: User has no documents available (and no special messages) after BFF call returns an empty list. The UI must handle this case clearly.
* **Missing Metadata**: A document is returned from the service without a description or title for the document name field. The UI should use available data or hide these fields gracefully.
* **Plan Type/Coverage Specifics (handled by client selecting correct policyResourceId/membershipResourceId):**
  + **Medicare Active, Pending different membershipIds:** Client makes separate calls for active and pending coverage using appropriate dateAsOf and membershipResourceId values.
  + **Medicare Active and Pending same membershipId.**
  + **IFP Active.**
  + **Commercial Active Medical, Dental, Vision.**
  + **Commercial Dependents Only Active Medical, Dental, Vision.**
  + **Commercial with both Active and Dependents Only Active for Dental** (Adult Dental/Pediatric Dental scenario).
  + **No plan docs enabled for any membership.**
* **PDF Retrieval:**
  + Successful retrieval and display of PDF.
  + Error during PDF retrieval (e.g., 0-byte PDF, network error) gracefully handled.

**Deployment Strategy (Brief)**

The new OpenPlatform BFF will be deployed as a microservice on the AWS platform, leveraging existing CI/CD pipelines and infrastructure-as-code practices. Deployments will be phased, starting with development and QA environments, followed by production.

**Rollback Plan (Brief)**

In the event of critical issues post-deployment, a rollback to the previous stable version of the OpenPlatform BFF will be initiated using automated deployment tools. The stateless nature of the BFF (with caching externalized) facilitates quicker rollbacks.

**Security Review**

A security review should be conducted, particularly as a new BFF is being introduced as an entry point into the AWS ecosystem. While the backend service patterns are being reused, the BFF's exposure, rate limiting, and authentication/authorization mechanisms need to be validated by the security team. This includes reviewing API security best practices, data encryption in transit and at rest, and access controls for the new BFF endpoints.

**Testing**

The solution should be tested to ensure all scenarios (zero, single, multiple documents) are handled correctly on the client and in the BFF. Testing should verify that missing metadata is handled gracefully and does not cause UI errors.

**Detailed Testing Plan**

To ensure robust functionality and a seamless user experience, the following detailed testing plan will be executed:

1. **Test Data Preparation:**
   * **User Profiles:** Create a diverse set of test users covering various scenarios:
     + **Zero Documents:** Users with no plan documents available (verify both Features API and BFF response paths).
     + **Single Document:** Users with exactly one plan document.
     + **Multiple Documents:** Users with several plan documents.
     + **Multi-language Users:** Users with documents available in different languages (e.g., English, Spanish), ensuring language field is correct.
     + **Users with Missing Metadata:** Users whose documents are known to have missing description or title fields from the upstream service.
   * **Plan Types & Coverage:**
     + **Medical-only:** Users covered only under a medical plan.
     + **Dental-only:** Users covered only under a dental plan.
     + **Vision-only:** Users covered only under a vision plan.
     + **Combined Plans:** Users with combinations (e.g., Medical + Dental, Medical + Vision, Medical + Dental + Vision), verifying correct policyResourceId and membershipResourceId selection by client.
     + **Varying Family Coverages:** Create scenarios where different members under the same plan have varied coverages:
       - **Example 1:** Plan holder has Medical, Dental, and Vision. Dependent children only have Medical and Dental (no Vision).
       - **Example 2:** Plan holder has Medical. Spouse has Medical and Dental.
       - **Example 3:** Different enrollment dates or policy periods impacting document availability for various family members.
       - Verify correct membershipResourceId and policyResourceId are used for each family member.
     + **HHL Not Signed Scenarios:** Test users associated with self-insured plans where the Hold Harmless Letter (HHL) has not been signed.
       - Verify client displays message when PlanDocsMsgAvail is enabled.
       - Verify client displays message when Benefits Service returns reasonCode: "HHL Not signed".
2. **Functional Testing:**
   * **End-to-End Flow:**
     + Verify the complete user journey from navigating to the Plan Documents section to viewing/downloading a PDF.
     + Confirm correct routing from the OpenPlatform client **through the Features API pre-check**, then through the new BFF to the existing backend services.
   * **Scenario Validation:**
     + **Features API Pre-check:**
       - Verify client correctly evaluates PlanDocsMsgAvail.
       - Verify client correctly evaluates DenPlanDocs, MedPlanDocs, VisionPlanDocs to determine if BFF call should proceed.
       - Confirm "Special Messaging" UI is displayed immediately when PlanDocsMsgAvail is enabled or no PlanDoc features are enabled.
     + **Zero Documents:** Verify the UI clearly communicates that no documents are available (after BFF call returns empty list).
     + **Single Document:** Confirm direct PDF opening behavior and proper PDF viewer display.
     + **Multiple Documents:** Validate accurate display of the document list, including names (derived from description/title and language) and descriptions (if present).
     + **Language Display:** Verify that language (e.g., "(Spanish)") is correctly appended to the document name in the list, based on the language field in the Benefits Service response.
     + **Missing Metadata Handling:** Confirm that description and title fields are handled gracefully (e.g., description is hidden if not provided; title used as fallback for documentName).
     + **PDF Actions:** Test "Share" and "Print" functionalities for PDF documents.
     + **reasonCode Handling:** Verify that if reasonCode: "HHL Not signed" is returned by the Benefits Service (via BFF), the client displays the appropriate special messaging, overriding any document display.
   * **Error Handling:**
     + Test various API error responses (e.g., 400 Bad Request, 403 Forbidden, 404 Not Found, 500 Internal Server Error, 503 Service Unavailable, 504 Timeout) from the Benefits Service and ensure the BFF translates and the client displays user-friendly messages.
     + Verify that 0-byte PDFs from core APIs are correctly handled (not cached, appropriate error returned).
3. **Performance Testing:**
   * **Latency Impact:** Measure response times for both list and PDF retrieval through the new BFF.
   * **Scalability:** Conduct load tests to ensure the BFF and backend services can handle anticipated OpenPlatform traffic volumes without degradation.
   * **Caching Effectiveness:** Monitor cache hit/miss ratios for Redis and S3 to confirm caching mechanisms are reducing load on core services as expected.
4. **Security Testing:**
   * Validate authentication and authorization mechanisms for the new BFF endpoints.
   * Conduct penetration testing to identify vulnerabilities.
   * Verify data encryption in transit (HTTPS) and at rest (for cached data in S3).

**Contacts and Impacted Teams/People**

* **CVS Contact**: Rohit Puri
* **Architecture**: Jenn Tang
* **Plan Doc Search**: Jesse Jackman
* **Product Management**: Darlene Scarola
* **AEI Team**: Wizards
* **Front-end Teams**: Wolf and Moonlight
* **Facets Contact**: Alan Chamberlain
* **Other Contacts**: Harshal Patel