**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 and will 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 calls the BFF, which in turn calls the backend Benefits Service to get the document list for that user. 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 |  |

Export to Sheets

**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.

* **List Retrieval**: The client calls 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 before sending it to the client.
* **PDF Retrieval**: The client requests a specific document via the BFF. 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.

**New OpenPlatform BFF Endpoints**

* **Get Document List**: POST /v1/cvs/plan-documents/list
  + **Description**: Retrieves the list of plan documents for the user.
  + **Request Body**: The client will send profile and plan identifiers.

JSON

{

"data": {

"id": "299814797",

"idType": "AETNA\_IM\_PROFILE\_ID\_TYPE",

"planId": "751133+BA+2"

}

}

* + **Response Body**: The response is a JSON object containing a data array of documents, tailored for the OpenPlatform.
    - **Zero Documents Example**:

JSON

{

"data":[]

}

* + - **Single Document Example**:

JSON

{

"data":[{

"planId" : "751133+BA+2",

"documentName":"Plan Document (Spanish)" ,

"documentId":"70023~13910995",

"documentUrl" :"/v1/cvs/plan-documents/retrieve"

}]

}

* + - **Multiple Documents Example**:

JSON

{

"data":[{

"planId" : "751133+BA+2",

"documentName":"Plan Document" ,

"documentId":"70023~13910995",

"documentUrl" :"/v1/cvs/plan-documents/retrieve"

}, {

"planId" : "751133+BA+2",

"documentName":"Plan Document (Spanish)" ,

"documentId":"70023~13910996",

"documentUrl" :"/v1/cvs/plan-documents/retrieve"

}]

}

* **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**:

JSON

{

"documentId": "70023~13910995"

}

* + **Response**: The raw PDF file (application/pdf).

**OpenAPI (YAML) Specification**

YAML

openapi: 3.0.3

info:

title: OpenPlatform - Plan Documents BFF

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

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:

- id

- idType

- planId

properties:

id:

type: string

example: '299814797'

idType:

type: string

example: 'AETNA\_IM\_PROFILE\_ID\_TYPE'

planId:

type: string

description: 'In Aetna this field is called policy id'

example: '751133+BA+2'

PlanDocListResponse:

type: object

properties:

data:

type: array

items:

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

DocumentMetadata:

type: object

properties:

planId:

type: string

example: '751133+BA+2'

documentName:

type: string

example: 'Plan Document (Spanish)'

documentId:

type: string

example: '70023~13910995'

documentUrl:

type: string

description: 'Nice to have'

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

PlanDocRetrieveRequest:

type: object

required:

- documentId

properties:

documentId:

type: string

description: 'Corresponds to communicationContentResourceId in the backend'

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.
  + Calls the backend Benefits Service's /plan-document-list/retrieve endpoint.
  + 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 to the document name for display purposes (e.g., "Plan Document (Spanish)").
    - Ensuring that if a description is not available, the field is not shown.
* **Benefits Service (Existing)**:
  + **List**: Logic remains as-is. It checks a Redis cache based on a complex key (considering Medicare vs. Commercial, ANOC, etc.) before calling the core API. The cache has a TTL of one hour.
  + **PDF**: Logic remains as-is. 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 user profile and plan, so that I can provide it to the OpenPlatform client.**
  + **Acceptance Criteria:**
    - The BFF successfully calls the /plan-document-list/retrieve endpoint on the Benefits Service.
    - The BFF handles various response scenarios from the Benefits Service (e.g., success, no documents, errors).
    - The BFF correctly transforms the Benefits Service response into the OpenPlatform API contract format (e.g., documentName with language, documentUrl).
    - The BFF gracefully handles cases where documentDescription or planName are missing from the upstream response by not including them in the final payload to the client.
* **As an OpenPlatform BFF, I want to retrieve a specific plan document PDF by its documentId, so that I can deliver it to the OpenPlatform client.**
  + **Acceptance Criteria:**
    - The BFF successfully calls the /plan-document/retrieve endpoint on the Benefits Service 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, so that I can reduce load on core APIs and improve response times.**
  + **Acceptance Criteria:**
    - The Benefits Service checks Redis for cached planDocListResponse based on a composite key (dateAsOf + policyResourceId + additional Medicare-specific identifiers).
    - 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 retrieved PDF; 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 for various scenarios.**
  + **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.
    - 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.
* **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.

* When the user accesses the feature, the client will call the BFF to get the list of documents.
* Based on the number of documents in the response array (data.length):
  + If 0, the client must handle the "zero document" scenario gracefully.
  + If 1, the client should directly open the PDF view upon the user's initial tap.
  + If >1, the client will display a list of the available documents for the user to choose from.
* 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 and there is only one plan document available, 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, 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 (including language, e.g., "Plan Document (Spanish)").
    - If a documentDescription is provided by the API, it is displayed for the corresponding document.
    - If documentDescription is *not* provided by the API, the field is not displayed.
    - 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, 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**

* The client will display the documentName from the API response for each document.
* A document description field will be displayed if it is present in the response; otherwise, it will be hidden.
* An attribute for language should be included to allow for filtering.
* The documentId will be sent back to the BFF to retrieve the specific PDF.

**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**

* **Single Document**: User has exactly one plan document. Tapping the link opens the PDF directly.
* **Multiple Documents**: User has more than one document. A list is displayed showing the title and description (if available).
* **Zero Documents**: User has no documents available. The UI must handle this case clearly.
* **Missing Metadata**: A document is returned from the service without a description or planName. The UI should hide these fields gracefully.
* **Multi-language**: A user has documents in multiple languages. The list should display the language, and the user should be able to filter by it.

**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. End-to-end testing from the OpenPlatform client through the new BFF to the existing backend services is required. Performance testing is required to ensure the new BFF does not introduce significant latency and that the overall experience does not degrade performance.

**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