**Feature Name: OpenPlatform - Plan Documents Access**

**Feature**

* **URL/Link to feature in Rally**: [F206620](https://www.google.com/search?q=https://rally1.rallydev.com/%23/%3Fdetail%3D/portfolioitem/feature/816688516123%26fdp%3Dtrue&authuser=1): Q1 Arch: Add communications content server side.

**Summary**

This feature will enable

CVS SuperApp users to access their plan documents. The project involves creating a new BFF (Backend-for-Frontend) for the CVS SuperApp, 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 CVS 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.

* **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 CVS 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.
  + **Pros**:
    - Reuses a proven, existing backend architecture and services.
    - Minimal impact on app startup time.
    - Leverages existing caching mechanisms in the Benefits Service and S3.
  + **Cons**:
    - Perceived performance on the Plan Documents page is dependent on a real-time API call.
    - Requires the BFF or client to handle potential data gaps (e.g., missing metadata) in the real-time response.
* **Option 2: App Startup Pre-fetch**
  + **Description**: Based on the observation that plan metadata is sometimes missing, this option proposes fetching the plan document list when the application starts up. 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.
  + **Pros**:
    - Extremely fast user experience when navigating to the Plan Documents page.
    - Allows for augmenting missing metadata (like PlanSponsorName) from a central source at a single point in time.
  + **Cons**:
    - Increases application startup time, which can negatively impact the overall user experience.
    - The pre-fetched data could become stale if changes occur during the user's session.
    - Increases complexity by introducing a new "app startup" data-fetching process.
* **Option 3: BFF-Centric Caching & Augmentation (Hybrid)**
  + **Description**: In this model, the CVS 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 SuperApp'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.
  + **Pros**:
    - Keeps app startup fast (like Option 1).
    - Provides a robust, centralized place (the BFF) to handle data inconsistencies and augmentation, simplifying client-side logic.
    - Caching can be optimized specifically for the CVS SuperApp's needs.
  + **Cons**:
    - Increases the complexity and statefulness of the BFF compared to a simple pass-through proxy.
    - May introduce slight latency compared to a pure pass-through, though this can be mitigated by caching.

**Chosen Direction**: **Option 1** is the chosen direction due to its simplicity and direct reuse of existing, proven infrastructure. To mitigate the risk of missing data, the CVS BFF will incorporate a limited augmentation capability (as described in Option 3) to normalize the data before it reaches the client.

**Solution Sketch**

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

1. **List Retrieval**: The client calls a new endpoint on the CVS 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.
2. **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**

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

1. CVS SuperApp -> CVS BFF
2. CVS BFF -> AWS APIc -> Benefits Service
3. Benefits Service -> Redis Cache (for list)
4. Benefits Service -> S3 Bucket (for PDF)
5. Benefits Service -> Core Plan Doc API (on cache miss)

**APIs/Swagger**

A new API contract will be defined for the CVS BFF based on the specific needs of the OpenPlatform.

**New CVS 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 SuperApp.
    - **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**

* **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 SuperApp traffic.
* **Future-proofing**: The architecture must not prevent the future implementation of search functionality.

**Services**

**Service Logic**

* **CVS BFF**:
  1. Receives a request from the SuperApp client for the document list.
  2. Calls the backend Benefits Service's /plan-document-list/retrieve endpoint.
  3. Receives the document list from the backend.
  4. Transforms the backend response into the format required by the SuperApp 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)**:
  1. **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.
  2. **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.

**Service Components**

* **New CVS BFF (Backend-for-Frontend)**: A new service specific to the CVS SuperApp, 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 CVS 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.

**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.

**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 SuperApp 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.

**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.

**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