CVS BFF Shared Documentation

Architecture

The following is meant to show the generalized architecture for the SuperApp BFF, use this diagram to avoid copying/pasting the same component in the child architectural diagrams, if possible.

Repo

https://github.com/cvs-health-source-code/aetnahealth-cvs-bff Connect your Github account

Cache Key

General usage:

<cvsbffPrefix>:route:<proxyResourceId> + (if available add impersonating LDAP)

1. Redis Cache Key Generation for Plan Groups

To optimize performance and reduce redundant backend calls, plan group responses are cached in Redis. The cache key is constructed using the following pattern:

```
cvsbff:plangroups:{proxyResourceId} + {impersonation LDAP aka idValue}
```

Components

- **Prefix:**
  `cvsbff:plangroups` — Identifies the cache as related to plan groups in the CVS BFF context.
- **proxyResourceId:**
  Extracted from:
  `eieheaderusercontext.eieHeaderBusinessIdentifier.businessIdentifier.find() => (idSource === 15).idValue`
- **idValue:**
  Extracted from the `accountIdentifier.idValue` field in the

`eieheaderusercontext` header.

ProTip - you can leverage localstorage helper methods from core-express to extract these. Example:

```
export const getCachePrefix = (): string => {
  const proxyResourceId = localStorageHelper.getProxyResourceId();
  const accountId = localStorageHelper.getAccountId();
  return `${config.CACHE_PREFIX}:${proxyResourceId}:${accountId}`;
};
```

Example

Given the following header:

```

"eieheaderusercontext": {
  "eieHeaderUserContext": {
    "accountIdentifier": {
      "idSource": "1",
      "idValue": "QA2-AYANAX",
      "idType": "accounts"
    }
    // ...other fields...
  }
}

```

example:

```
cvsbff:plangroups:MXN76FFFFPXZ:QA2-AYANAX
```

Cache Key Generation Documentation

Overview

The cache system in the CVS BFF service uses Redis to store frequently accessed data. Cache keys are dynamically generated to ensure proper data isolation between users and impersonation scenarios while maintaining efficient cache lookup.

Cache Key Structure

Cache keys follow this hierarchical structure:

```
{CACHE_PREFIX}:{domain}:{accountKey}:{uniqKeyIdentifier}

```

Components Breakdown

1. Cache Prefix (`CACHE_PREFIX`)

- **Source**: Environment configuration (`config.CACHE_PREFIX`)
- **Purpose**: Namespace isolation to prevent key collisions across different environments or services
- **Example**: `cvs-bff:dev` or `cvs-bff:prod`

2. Domain

- **Source**: Predefined constants from `CacheDomains`
- **Purpose**: Categorizes cached data by functionality
- **Available Domains**:
  - `idCardsList` - Cached ID cards list data
  - `idCardsDetails` - Detailed ID card information
  - `memberDetails` - Member-specific details
  - `planGroups` - Plan group information

3. Account Key (`accountKey`)

- **Composition**: `${accountId}${impersonatingAccountId}`
- **Components**:
  - `accountId` : Retrieved from local storage helper (`localStorageHelper.getAccountId()`)
  - `impersonatingAccountId` : Extracted from EIE headers using `getImpersonatingAccountIdFromHeaders()`
- **Purpose**: Ensures data isolation between different users and handles impersonation scenarios

4. Unique Key Identifier (Optional)

- **Source**: Parameter passed to `buildCacheKey()` function
- **Purpose**: Further differentiate cache entries within the same domain and account
- **Examples**: Member IDs, plan IDs, or other business-specific identifiers

Key Generation Process

Step-by-Step Flow

1. **Retrieve Account Information**

```
const accountId = localStorageHelper.getAccountId();
```

```
```

2. **Extract EIE Headers**

```
const headers = localStorageHelper.getEieHeaders();
const parsedHeaders = parseEieHeaders(headers);

```

3. **Handle Impersonation**

```
const impersonatingAccountId = getImpersonatingAccountIdFromHeaders(parsedHeaders);

```

4. **Build Account Key**

```
const accountKey = `${accountId}${impersonatingAccountId}`;

```

5. **Construct Final Cache Key**

```
return `${config.CACHE_PREFIX}:${domain}:${accountKey}${uniqKeyIndentifier ? `:${uniqKeyIndentifier}` : ''}`;

```

Example Cache Keys

Standard User Access

```
cvs-bff:dev:idCardsList:12345
cvs-bff:dev:memberDetails:12345:member-67890
cvs-bff:dev:planGroups:12345

```

Impersonation Scenario

When a user with account ID `admin123` is impersonating account `12345`:

```
cvs-bff:dev:idCardsList:12345admin123
cvs-bff:dev:memberDetails:12345admin123:member-67890

```

Domain-Specific Examples

Domain	Example Key	Description
`idCardsList`	`cvs-bff:prod:idCardsList:12345`	List of ID cards for account 12345

`idCardsDetails`	`cvs-bff:prod:idCardsDetails:12345:card-abc123`	Detailed info for specific card
`memberDetails`	`cvs-bff:prod:memberDetails:12345:member-xyz789`	Details for specific member
`planGroups`	`cvs-bff:prod:planGroups:12345`	Plan groups for account 12345

Cache Operations

Setting Cache Values

```
await setCacheValue({
  domain: CacheDomains.MEMBER_DETAILS,
  uniqKeyIndentifier: 'member-12345',
  data: memberData,
  cacheTTL: 3600 // Optional TTL override
});

```

Getting Cache Values

```
const cachedData = await getCacheValue<MemberDetailsType>(
  CacheDomains.MEMBER_DETAILS,
  'member-12345'
);

```

Security Considerations

Data Isolation

- **Account-based isolation**: Each account's data is cached separately
- **Impersonation handling**: When users impersonate others, cache keys include both original and impersonating account IDs
- **Domain separation**: Different data types are isolated by domain

EIE Header Integration

- **Header parsing**: Uses `@aetnadigital/eie_header_utils` for secure header processing
- **Impersonation detection**: Automatically detects and handles impersonation scenarios
- **Context preservation**: Maintains user context across cache operations

Configuration

Environment Variables

- `CACHE_PREFIX`: Sets the namespace prefix for all cache keys
- `CACHE_TTL`: Default time-to-live for cached entries (in seconds)
- `DISABLE_CACHE`: Boolean flag to disable caching entirely

Cache Domains

Cache domains are defined as constants to prevent typos and ensure consistency:

```
export const CacheDomains = {
  ID_CARDS_LIST: 'idCardsList',
  ID_CARDS_DETAILS: 'idCardsDetails',
  MEMBER_DETAILS: 'memberDetails',
  PLAN_GROUPS: 'planGroups',
} as const;

```

Best Practices

Key Generation

1. **Always use predefined domains** from `CacheDomains` constants
2. **Include unique identifiers** when caching entity-specific data
3. **Consider impersonation scenarios** - the system handles this automatically
4. **Use meaningful unique identifiers** that clearly identify the cached data

Cache Management

1. **Set appropriate TTL values** based on data volatility
2. **Handle cache misses gracefully** - always have fallback logic
3. **Monitor cache hit rates** to optimize key strategies
4. **Use consistent naming patterns** for unique identifiers

Error Handling

- Cache operations are wrapped in try-catch blocks
- Failed cache operations log warnings but don't break application flow
- Cache disabled mode returns `undefined` for all get operations

Troubleshooting

Common Issues

1. **Cache Misses Due to Key Mismatch**
    - Verify account ID retrieval is consistent
    - Check EIE header parsing
    - Ensure unique identifiers match between set/get operations
2. **Impersonation Context Loss**
    - Verify EIE headers are properly set in local storage
    - Check impersonation account ID extraction logic
3. **Environment Isolation Issues**
    - Confirm `CACHE_PREFIX` is environment-specific
    - Verify Redis instance separation between environments

Debug Cache Keys

To debug cache key generation, you can temporarily log the generated keys:

```
const cacheKey = buildCacheKey(domain, uniqKeyIndentifier);
logger.debug(`Generated cache key: ${cacheKey}`);

```

Related Documentation

- EIE Header Utils Documentation
- Redis Configuration Guide
- Environment Configuration

URL Convention

url standard:

/sa/:domain/:version/:(optionalresource)

example:

/sa/idcards/v1/list

Application Startup Flow

The `CVS BFF` routes orchestrate the retrieval of member Data cards by leveraging three key backend APIs in sequence. This flow ensures that only eligible members and plans are considered, and that feature flags and member details are respected.

1. Plan Groups API (`/plangroups`)

Purpose:
Fetches all plan groups associated with the current user. This provides the foundational data set of plans and memberships to be filtered and processed.

General Use:

To obtain a unique plan group, you can query the API using the following combination:
- lineOfBusiness + relationshipToSubscriber + planSponsorId

Request Example:

```
GET /v3/plangroups HTTP/1.1
Host: plan-groups-service.dev.aetnadigital.net
Authorization: Bearer <token>
Content-Type: application/json
```

Response Example:

```
{
  "planGroups": [
    {
      "lineOfBusinessName": "Commercial",
      "planSponsorId": "0000000086517890",
      "planSponsorName": "DMT-K-MURU-0012-01",
      "relationshipToSubscriber": "Other Relative",
      "policies": [
        {
          "primaryPolicyType": "Medical",
          "memberships": [
            {
              "status": "Actively Covered",
              "relationshipToSubscriber": "Other Relative",
              "memberResourceId": "41~266119007",
              "membershipResourceId": "5~266119007+10+1+20220101+808021+A+1",
              "effectiveDatetimeBegin": "2022-01-01T04:00:00Z",
              "effectiveDatetimeEnd": "9999-12-31T04:00:00Z"
            }
          ]
        }
      ]
    }
  ]
}
```

2. Multi-Membership Features API (`/multimembership features`)

Purpose:

Determines which memberships have specific features enabled (e.g., `CVSSuperApp`). This step filters memberships to only those eligible for ID card retrieval based on feature flags.

Request Example:

```
POST /v2/features HTTP/1.1
Host: multimembershipfeatures.dev.aetnadigital.net
Content-Type: application/json

{
  "enabled": true // optional
  "memberships": [
    {
      "membershipResourceId": "5~266119007+10+1+20220101+808021+A+1",
      "features": ["CVSSuperApp"]
    }
  ]
}
```

Response Example:

```
{
  "features": [
    {
      "code": "CVSSuperApp",
      "memberships": [
        {
          "membershipResourceId": "5~266119007+10+1+20220101+808021+A+1",
          "status": "ONLINE",
          "enabled": true
        }
      ]
    }
  ]
}
```

3. Member Details API (`/ivl/memberships Memberships`)

Purpose:

Provides member data specific to a given membershipResourceId

Request Example:

```
GET /v1/memberships/:membershipResourceId HTTP/1.1
Host: coreproxy.dev.aetnadigital.net
Content-Type: application/json
```

```
 
```

Response Example:

```
{
  "membershipResponse": {
    "membershipDetail": {
      "privacyRestriction": false,
      "relationshipToSubscriber": "Self",
      "relationshipToSubscriberCode": "18",
      "status": "Actively Covered",
      "memberId": "300003556500",
      "idCardText": "300003556500",
      "memberResourceId": "81~300003556500",
      "primaryInsuranceFlag": false,
      "person": {
        "nameFirst": "Ayanax",
        "nameMiddle": "A",
        "nameLast": "Guillory",
        "gender": "F",
        "nameFull": "Ayanax A Guillory",
        "dateOfBirth": "1933-01-01",
        "tobaccoUse": "unknown",
        "contacts": [
          {
            "usage": "personal",
            "postalAddresses": [
              {
                "state": "TX",
                "postalCode": "77002",
                "streetLine1": "1500 HADLEY ST",
                "city": "HOUSTON",
                "countyCode": "48201",
                "country": {
                  "code": "US"
                }
              }
            ]
          },
          {
            "usage": "home",
            "postalAddresses": [
              {
                "state": "TX",
                "postalCode": "77002",
                "streetLine1": "1500 HADLEY ST",
                "city": "HOUSTON",
                "countyCode": "48201",
                "country": {
                  "code": "US"
                }
              }
            ]
          },
          {
            "primaryPhone": {
              "number": "8556473504",
              "type": "Primary"
          },
          "emailAddresses": [
            {
              "address": "nextgentestdatamanagement",
              "domain": "aetna.com"
            }
          ]
        }
      ]
    },
    "enrollmentSourceCode": "B",
    "applicationReceivedDate": "2019-11-24",
    "outOfServiceArea": false,
    "medicareMembership": {
      "applicationSignatureDate": "2019-11-24",
      "applicationSubmitDate": "2019-11-24",
      "applicationCompletionDate": "2019-11-25",
      "enrollmentPeriodElectionTypeCode": "E-IEP",
      "partDRxProcessorControlNumber": "MEDDAET",
      "partDRxBankIdentificationNumber": "610502",
      "partDRxGroupNumber": "RXAETD"
    },
    "medicareIdentifier": [
      {
        "code": "MBI",
        "valueShort": "2G11M36RA15",
        "effectivePeriod": {
          "datetimeAsOf": "2020-03-17T12:23:57",
          "datetimeBegin": "2019-11-01T00:00:00-04:00",
          "datetimeEnd": "3000-01-01T00:00:00"
        },
        "medicareMembershipHistory": [
          {
            "dataElementName": "Medicare Part A",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Medicare Part B",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Medicare Part D",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Part D LIS Copay Category",
            "dataElementCodeValue": "0",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Part D LIS Subsidy Level",
            "dataElementCodeValue": "000",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Part D Current LIS Premium Subsidy",
            "dataElementDoubleValue": 0,
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Premium Withhold Payment Method Type",
            "dataElementDescriptionValue": "Direct Pay",
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Part D Creditable Coverage Indicator",
            "dataElementDescriptionValue": "true",
            "effectivePeriod": {}
          },
          {
            "dataElementName": "Part D LEP Current Number of Uncovered Months",
            "dataElementDescriptionValue": "0",
            "dataElementDoubleValue": 0,
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
          },
          {
            "dataElementName": "Part D LEP Total Number of Uncovered Months",
            "dataElementDescriptionValue": "0",
            "dataElementDoubleValue": 0,
            "effectivePeriod": {
              "datetimeAsOf": "2020-03-17T12:23:57",
              "datetimeBegin": "2020-01-01T00:00:00-05:00",
              "datetimeEnd": "3000-01-01T00:00:00"
            }
            }
          ]
        }
      ],
      "producer": {
        "nationalProducerNumber": ""
      },
      "coverageBillingTier": "Individual",
      "alternateIdentifications": [
        {
          "code": "IDT52",
          "valueShort": "N00085031816"
        },
        {
          "code": "CN",
          "valueShort": "BC56421415"
        }
      ],
      "enrollment": {
        "primaryPolicyType": "Medical",
        "policyCategory": "PPO",
        "policyIdentifier": {
          "idSource": "68",
          "idValue": "000003-TX000003",
          "idType": "healthpolicies",
          "resourceId": "68~000003-TX000003"
        },
        "lineOfBusinessCode": "M"
      },
      "membershipIdentifier": {
        "idSource": "59",
        "idValue": "300003556500+000003-TX000003+2020-01-01",
        "idType": "memberships",
        "resourceId": "59~300003556500+000003-TX000003+2020-01-01"
      },
      "individual": {
        "individualIdentifier": {
          "idSource": "15",
          "idValue": "MXN76FFFFPXZ",
          "idType": "individuals",
          "resourceId": "15~MXN76FFFFPXZ"
        }
      },
      "effectivePeriod": {
        "datetimeBegin": "2020-01-01T00:00:00",
        "datetimeEnd": "3000-01-01T00:00:00",
        "datetimeAsOf": "2020-03-17T12:23:57"
      },
      "rxCAGs": [
        {
          "rxCAGIdentifier": {
            "idSource": "130002",
            "idValue": "313679258",
            "idType": "CAGS",
            "resourceId": "130002~313679258"
          },
          "effectivePeriod": {
            "dateAsOf": "2020-03-17",
            "dateBegin": "2020-01-01",
            "dateEnd": "3000-01-01"
          }
        }
      ]
    }
  }
}
```

CVS Documentation

- Status Codes and Descriptions → Status Codes
- Open Platform Docs → Open Platform - Plug & Play