**1) Contract governance & versioning**

* **OpenAPI-first**: Treat OpenAPI as the source of truth. Gate merges on spectral lint + breaking-change checks; generate clients/DTOs.
* **Path versioning**: Adopt /sa/:domain/:version/... (e.g., /sa/idcards/v1/list) with explicit deprecation windows and canary consumers. Keep **idempotency** of GET/HEAD and standard RFC 7231 verbs.
* **Release channels**: vNext behind feature flags, publish change logs, provide parallel **/vN** and **/vN+1** for migration.

**2) Authorization & token passing**

* Centralize an **AuthProvider** with pluggable strategies (OAuth2 CC/JWT bearer, mTLS, API key when forced). Token cache keyed by (client\_id, scope(s), audience), eager refresh at 80% TTL, clock-skew safe.
* **Header policy**: Always stamp correlation/transaction IDs + user context headers (EIE family) on egress; never leak raw user tokens externally.
* **Impersonation**: Treat impersonated identity as part of the call context and cache keys; never blend results across real vs impersonated identities.

**3) Request envelope, idempotency & concurrency control**

* **Correlation**: Require a traceparent (W3C) and a business refId/transaction id on every call.
* **Idempotency keys** for POST/PUT with safe retries; partner-specific replay windows captured in policy.
* **Time boxing**: Distinct connect/TLS/read timeouts; hard ceiling per call via deadline propagation.

**4) Error normalization & partner mapping**

* Normalize to **RFC 7807** (problem+json). Preserve **raw partner payload** in structured telemetry.
* Classify partner errors into **Retryable | NonRetryable | CallerError | PolicyFiltered**; attach partner.status\_code, partner.operation, partner.correlation\_id.
* Ship a registry of partner → normalized error codes with test coverage.

**5) Resilience patterns**

* **Retries**: Exponential backoff + jitter (retry only idempotent operations or POSTs with idempotency keys).
* **Circuit breakers** with rolling error budgets; **bulkheads** per partner domain.
* **Hedging** (optional) for high-latency GETs against read replicas when offered.

**6) Data minimization & privacy**

* Only send the attributes required by the contract; redact secrets/PII in logs by default.
* Enforce **policy filters** (e.g., coverage windows, entitlements) *before* cache write and response mapping.

**7) Caching strategy (client & server side)**

* **HTTP-aware**: Support ETag/If-None-Match when partners expose them; otherwise application caching.
* **App-layer Redis** with **namespaced keys** and impersonation isolation. Pattern:
* {CACHE\_PREFIX}:{domain}:{accountKey}:{uniqKey}
* accountKey := accountId + impersonatingAccountId

Use stable prefixes per environment (e.g., cvs-bff:dev|prod) and include per-request unique identifiers (membership/plan IDs).

* When a proxy resource id exists, include it to avoid cross-tenant bleed (e.g., cvsbff:plangroups:{proxyResourceId}:{idValue}).
* TTLs by data volatility; support **stale-while-revalidate** on cache miss bursts.

**8) Pagination, filtering, and partial responses**

* Standardize pageSize (hard cap), pageToken, filter, sort, and field-masking (sparse fieldsets) to control payload size/cost.

**9) Observability & SLOs**

* **Traces**: Propagate context across all egress calls; annotate spans with partner, route, cache.hit, impersonation.
* **Metrics**: p95/99 latency per operation; success/error rates; cache hit ratio; retry counts; policy-filtered counts (e.g., “out-of-policy dropped”).
* **Logs**: Sampled structured logs with claim/resource identifiers hashed; never log raw tokens.

**10) Security controls**

* **mTLS** to partner where possible; egress allow-listing; DNS pinning or cert pinning for high-risk partners.
* **Secrets** from KMS/SM; automatic rotation; zero tokens in code/config.
* SSRF protections on any partner-provided URLs; strict content-type & size limits.

**11) SDK “Outbound Connector” (shared component)**

* A single library used by all services to call partners; provides:
  + Request building (headers, correlation IDs, EIE context) and URL convention helpers.
  + AuthProvider injection.
  + Retry/circuit logic and error normalization.
  + Cache adapter (Redis) with **account + impersonation** keying.
  + Metrics/tracing interceptors.

**Minimal TypeScript shape**

export interface PartnerConnectorOpts {

baseUrl: string;

operation: string; // e.g., "plangroups.get"

cacheDomain?: string; // e.g., "planGroups"

uniqueKey?: string; // e.g., membershipId

idempotencyKey?: string;

auth: AuthProvider;

}

export async function callPartner<TReq, TRes>(opts: PartnerConnectorOpts, req: TReq): Promise<TRes> {

const ctx = buildContext(); // traceparent, refId, EIE headers

const hdrs = await opts.auth.headers(); // bearer/mTLS/api key

const cacheKey = buildCacheKey(opts.cacheDomain, ctx.accountId, ctx.impersonatingId, opts.uniqueKey);

const cached = await cache.get<TRes>(cacheKey);

if (cached) return cached;

const res = await httpClient(opts.baseUrl)

.withTimeouts({ connect: 500, total: 3000 })

.withHeaders({ ...hdrs, ...ctx.headers })

.withRetries({ attempts: 3, backoff: "exp-jitter", idempotent: !!opts.idempotencyKey })

.post<TRes>(routeFor(opts.operation), req);

const normalized = normalizePartnerResponse(res);

await cache.put(cacheKey, normalized, ttlFor(opts.cacheDomain));

return normalized;

}

* buildCacheKey should follow {CACHE\_PREFIX}:{domain}:{accountId}{impersonatingId}:{uniqKey}, honoring the environment prefix and EIE extraction utilities.

**12) API composition order of operations (reference)**

1. **Auth headers** + **EIE context** stamped.
2. **Cache read** using namespaced, impersonation-aware key.
3. **Partner call** with timeouts/retries/circuit protection.
4. **Policy filters** (eligibility windows, feature flags) before mapping & cache write.
5. **Mapping** to internal DTOs; redact; **cache write**.
6. **Error normalization** to RFC 7807 + metrics/traces.

**13) Governance & CI/CD hooks**

* **Lint/validate** every OpenAPI; generate clients; publish versioned SDK.
* **Contract tests** against partner sandbox + **synthetic probes** in prod.
* **Feature flags** for partner endpoints, cache TTLs, and policy filters (dry-run vs enforce).

**Pull-throughs from current BFF patterns**

* **Cache keys**: Use environment prefixes and concatenate account + impersonation IDs; domain-scoped keys (e.g., planGroups, memberDetails).
* **EIE header utilities**: Centralize header parsing and impersonation detection; preserve user context in cache and outbound calls.
* **URL convention**: Keep /sa/:domain/:version for consumer clarity and change i