■ AI Prompt — Convert **Client Credentials** flow to **Authz Flows v3** (code-aware, reuse-first)

Goal: Implement/upgrade the **Client Credentials** flow to fully match **Authz Flows v3** styling, copy, step structure, logging, and hardening—**reusing existing components, hooks, and utils**. No duplicated logic. Ship behind a feature flag with full tests and observability.

> Notes: > - Client Credentials is **machine-to-machine** (no browser redirect). We **only** obtain and manage an **access_token** (optionally token_type, expires_in, scope). > - Support **client_secret_basic**, **client_secret_post**, and **private_key_jwt** (preferred). > - Provide optional **mTLS** toggle if the repo already supports it in HTTP layer.

---

0) Guardrails & Parity

- **UX parity with V3:** same layout, colors, stepper, buttons, spinners, toasts, tooltips, and copy tone. - **Reuse-first:** extract common parts to `src/utils/*` instead of cloning. - **Unified logging:** use `src/utils/logger.ts` with emoji/module tags. - **Config resolution order:** `.env` → `settings.json` → `localStorage` via `src/services/config` and `src/utils/credentialManager`. - **Hardening:** strict validation, typed APIs, graceful error surfaces, consistent error cards/toasts.

---

1) Files to (Re)use & Where to Plug In

- **Context / Session** - `src/contexts/NewAuthContext.tsx` (expose M2M token fetch/use; token presence helpers) - **Flow Pages** - Start/Flow page: `src/pages/flows/ClientCredentialsFlow.tsx` - Optional results/status panel: reuse `src/components/TokenDisplay.tsx` (access token only) - Use shared UI: `src/components/StepByStepFlow.tsx`, `src/components/PageTitle.tsx`, `src/components/ConfigurationButton.tsx`, `src/components/ColorCodedURL.tsx` - **Config & Discovery** - `src/services/config` (env + settings merge) - `src/services/discoveryService.ts` (read `token_endpoint`) - `src/config/pingone.ts` (PingOne helpers if present) - **HTTP & OAuth Utils** - `src/utils/oauth.ts` (extend with CC helpers) - `src/utils/tokenStorage.ts`, `src/utils/storage.ts` (persist token + expiry) - `src/utils/tokenLifecycle.ts`, `src/utils/tokenHistory.ts` (status/expiry/history) - `src/utils/urlValidation.ts`, `src/utils/secureJson.ts` - `src/utils/logger.ts` (required) - **Types** - `src/types/*` for oauth/auth/storage/errors

---

2) Routes & Navigation

- Register routes: - **Start**: `/flows/client-credentials` → `ClientCredentialsFlow.tsx` - This flow is back-channel only (no `/callback` route).

---

3) Functional Spec

3.1 Start / Configure (`/flows/client-credentials`) - Required inputs (validate like V3): - `token_endpoint`, `client_id`, `auth_method` (one of: `client_secret_basic`, `client_secret_post`, `private_key_jwt`), `scope` (optional if audience-based), optional `audience`/`resource` (PingOne APIs), optional `tenant/env` selectors. - Defaults & advanced: - `grant_type = client_credentials` - Toggle `private_key_jwt` (preferred); if enabled, collect `issuer`, `subject`, `aud` (token_endpoint), `kid`, and reference `privateKey` from `credentialManager`. - Optional `mtls.enabled` & cert refs if supported by your HTTP layer.

- Build request **without redirect**: - For `client_secret_basic`: set `Authorization: Basic base64(client_id:client_secret)`. - For `client_secret_post`: include `client_id` & `client_secret` in body. - For `private_key_jwt`: sign JWT: - `iss=sub=client_id`, `aud=token_endpoint`, `jti` random, `iat/exp` (≤60s), header `kid`, alg per key (`RS256`/`ES256`), param `client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer`, `client_assertion=<jwt>`. - Body params: `grant_type=client_credentials`, `scope` (space-delimited) OR `resource/audience` as required.

- **Log examples** - `[■■ CC] Preparing client credentials request (auth_method=${auth_method})` - `[■ CC] Using credential source=${source}` (env/settings/localStorage/keystore)

3.2 Token Request - Send POST to `token_endpoint` with `application/x-www-form-urlencoded`. - Handle HTTP, network, and OAuth error bodies (JSON). - On success, persist: `access_token`, `token_type`, `expires_in`, `scope?`, absolute `expAt`.

*3.3 Post-Token UX - Token status panel (same style as V3): - Show **access token** (masked) with copy button, TTL countdown, and history entry. - Provide a small **Resource Call** demo (optional): reuse existing `ApiClient` to call a sample PingOne endpoint when `audience/resource` indicates one; show status + response body snippet. - Status bar shows env ID, region, version, and token presence—parity with V3.*

---

4) Security & Hardening

- Prefer **private_key_jwt** over shared secrets; enforce strong key length/alg. - If secrets are used, mask in UI; never log secrets/keys. Redact in errors. - Strict input validation (URLs, scopes, audiences). Use `urlValidation`. - Short-lived client assertions (≤60s); unique `jti`; clock-skew tolerance (±2–5 min). - `tokenStorage`: store only what is necessary (no id_token); purge on sign-out. - Optional **mTLS** (if available): attach client cert/key via HTTP layer config. - Least-privilege scopes; support `audience/resource` to constrain token. - Retries: exponential backoff on transient 5xx, **no** retry on 4xx invalid_client. - Feature flag: `config.oauth.clientCredentials.enabled` (default off in prod).

---

5) Reuse-First Refactors (concrete)

Create/extend helpers in `src/utils/`:

- `buildClientCredentialsBody(opts): URLSearchParams` - `signClientAssertion({ clientId, tokenEndpoint, kid, privateKey, alg }): string` (via `jose`) - `requestClientCredentialsToken({ discovery, authMethod, body, headers, mtls? }): Promise<TokenResponse>` - `computeAbsoluteExpiry(expires_in): number` - `TokenCache` keyed by `{audience|resource, scope, client_id, env}` with proactive refresh threshold (e.g., 60–120s before expiry) - Reuse `tokenStorage.put/get/clear('client_credentials')` - Reuse `tokenLifecycle` for countdown/status bar

> Rule: if new code is ≥70% similar to V3 utilities, **extract** and parametrize.

---

6) Telemetry & Logging

Use `logger` consistently:

- Build: `[■■ CC] body built scopes="${scopes}" audience="${audience||''}"` - Request start: `[■ CC] POST /token method=${auth_method}` - Success: `[■ CC] token acquired exp=${expIso} token_type=${token_type}` - Cache: `[■ CC] cache hit ttl=${ttl}s` / `[■■ CC] proactive refresh in ${n}s` - Error: `[■ CC] token request failed code=${oauth_error} http=${status}`

All logs emoji'd, timestamped, module-tagged, non-blocking, redacting secrets.

---

7) Configuration Additions

Augment `settings.json` (respect `.env` → `settings.json` → `localStorage`) via `services/config`:

```json { "oauth": { "clientCredentials": { "enabled": true, "authMethod": "private_key_jwt", "scopes": ["p1:read:users"], "audience": "", "resource": "", "proactiveRefreshSeconds": 90, "assertion": { "alg": "RS256", "kid": "YOUR_KID" }, "mtls": { "enabled": false, "certRef": "", "keyRef": "" } } } } ```

Secrets/keys should come from the **Credential Manager** and never be hard-coded.
```

---

8) Test Plan

- **Unit** - `buildClientCredentialsBody` builds correct bodies for each auth_method. - `signClientAssertion` signs correct header/payload; rejects stale `iat/exp`. - `requestClientCredentialsToken` handles 2xx JSON, 4xx/5xx error bodies. - `TokenCache` returns cached token and triggers proactive refresh correctly. - **Integration** - Mock token endpoint; verify success path, secret masking, error surfaces. - Negative cases: invalid_client, invalid_grant, unsupported_grant_type, auth mismatch. - **E2E** - `/flows/client-credentials` → “Get Token” → status bar shows token; optional sample API call succeeds. - Proactive refresh occurs without UI freeze. - **Accessibility** - Parity with V3: focus order, aria labels, keyboard nav.

---

9) Acceptance Criteria (ship checklist)

- [ ] **Exact** styling/copy parity with V3 flow pages/components. - [ ] No duplicated business logic; shared utils in `src/utils/*`. - [ ] Supports `client_secret_basic`, `client_secret_post`, **`private_key_jwt`**. - [ ] Secrets/keys never logged; UI masking + redaction enforced. - [ ] Access token stored with absolute expiry; status bar countdown present. - [ ] Proactive refresh honored; cache keyed by scope/audience/client/env. - [ ] Feature flag `oauth.clientCredentials.enabled` gates UI/route. - [ ] Unit + integration + E2E tests green; coverage ≥ V3 baseline.

Implement exactly as specified using existing repo patterns—`NewAuthContext`, `StepByStepFlow`, `tokenStorage`, `discoveryService`, `logger`, and your HTTP client—so the Client Credentials flow “feels” like Authz v3 and stays maintainable.