Awesome—here's a clean, implementation-ready design for **two token endpoints** you can drop into your C2M API v2 spec and Postman. It uses an **exchange** model:

- **Short-term token** (JWT, ~15 minutes) is minted **by presenting a valid long-term token**.
- **Long-term token** (~30–90 days, opaque or JWT) is minted via a **separate call** that requires step-up auth (client credentials + one of: OTP, signed assertion, or an existing short token).

Below is an OpenAPI 3.0.3 snippet you can paste into your spec. I used neutral names and clear lifetimes; tweak durations, header names, and flows to your liking.

✅ Endpoints overview

- **`POST /auth/tokens/short`** — Exchange a **long-term token** for a **short-term token**
- **`POST /auth/tokens/long`** — Create/rotate a **long-term token** (requires step-up)

Both return token metadata (type, **`expires_in`**, **`expires_at`**, **`scopes`**) and support **revocation** via **`POST /auth/tokens/{tokenId}/revoke`** (optional, included below for completeness).

🔧 OpenAPI YAML (drop-in)

```yaml
yaml


openapi: 3.0.3
info:
  title: C2M API v2 - Auth
  version: 2.0.0

paths:
  /auth/tokens/short:
    post:
      summary: Issue a short-term access token
      description: |
        Exchanges a valid **long-term token** for a short-lived access token (e.g., 15 minutes).
        Use this token to call operational endpoints. The long-term token must be sent either
        in the Authorization header as `Bearer <long_token>` or in the request body.
      security:
        - LongTokenAuth: []   # see components/securitySchemes
      requestBody:
        required: false
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ShortTokenRequest'
            examples:
              withScopes:
                value:
                  scopes: ["jobs:submit", "templates:read"]
      responses:
        '201':
          description: Short-term token issued
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ShortTokenResponse'
              examples:
                ok:
                  value:
                    token_type: Bearer
                    access_token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                    expires_in: 900
```

```
                  expires_at: "2025-08-26T22:45:00Z"
                  scopes: ["jobs:submit","templates:read"]
                  token_id: "tok_st_9f3d2c"
        '400':
          description: Invalid request
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '401':
          description: Long-term token missing or invalid
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '429':
          description: Rate limited
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }

  /auth/tokens/long:
    post:
      summary: Issue or rotate a long-term token
      description: |
        Issues a **long-term token** (e.g., 30–90 days) for automation or server-to-server use.
        Requires step-up authentication. Choose one of the supported grant types:

        - `client_credentials` + `client_secret`
        - `client_credentials` + one-time `otp_code`
        - `assertion` (e.g., signed JWT by your private key)
        - Or an existing short-term token with the `tokens:write` scope

        Only one mechanism is required per call.
      security:
        - ClientKey: []           # optional if you pass body credentials
        - ShortTokenAuth: [ "tokens:write" ]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/LongTokenRequest'
            examples:
              clientSecret:
                value:
                  grant_type: client_credentials
                  client_id: "c2m_abc123"
                  client_secret: "•••"
                  scopes: ["jobs:submit","templates:read"]
                  ttl_seconds: 7776000    # 90 days
              otp:
                value:
                  grant_type: client_credentials
                  client_id: "c2m_abc123"
                  otp_code: "123456"
                  scopes: ["jobs:*"]
                  ttl_seconds: 2592000     # 30 days
              assertion:
                value:
                  grant_type: assertion
                  client_id: "c2m_abc123"
                  assertion_type: "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
                  assertion: "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9..."
                  scopes: ["jobs:submit","templates:read"]
      responses:
        '201':
          description: Long-term token issued
          headers:
            X-Token-Id:
              description: Server-generated identifier for revocation and audit
              schema: { type: string }
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/LongTokenResponse'
              examples:
```

```yaml
                ok:
                  value:
                    token_type: Bearer
                    access_token: "lt_0d51e9bd2f6449cf..."
                    expires_in: 2592000
                    expires_at: "2025-09-25T22:45:00Z"
                    scopes: ["jobs:*"]
                    token_id: "tok_lt_b8e0aa"
        '400':
          description: Invalid request
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '401':
          description: Authentication failed
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '403':
          description: Caller lacks tokens:write or policy disallows issuance
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '429':
          description: Rate limited
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }

  /auth/tokens/{tokenId}/revoke:
    post:
      summary: Revoke a token
      description: Revokes a short- or long-term token by its `tokenId`.
      security:
        - ShortTokenAuth: [ "tokens:revoke" ]
        - LongTokenAuth:  [ "tokens:revoke" ]
      parameters:
        - name: tokenId
          in: path
          required: true
          schema: { type: string }
      responses:
        '204':
          description: Revoked (idempotent)
        '401':
          description: Auth failed
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }
        '404':
          description: Token not found (already revoked or unknown)
          content:
            application/json:
              schema: { $ref: '#/components/schemas/Error' }

components:
  securitySchemes:
    # Use this if callers send the long-term token in Authorization header
    LongTokenAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

    # Use this if callers send a short-term token for step-up and revocation
    ShortTokenAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

    # Optional: client key in header if you require it globally
    ClientKey:
      type: apiKey
      in: header
      name: X-Client-Id
```

```yaml
  schemas:
    ShortTokenRequest:
      type: object
      properties:
        long_token:
          type: string
          description: Optional if the long-term token is provided in Authorization header
        scopes:
          type: array
          items: { type: string }
          description: Optional scope narrowing; defaults to the long-term token's scopes
      additionalProperties: false

    ShortTokenResponse:
      type: object
      required: [ token_type, access_token, expires_in, expires_at ]
      properties:
        token_type:
          type: string
          enum: [ Bearer ]
        access_token:
          type: string
          description: Short-lived JWT
        expires_in:
          type: integer
          description: Lifetime in seconds (e.g., 900)
        expires_at:
          type: string
          format: date-time
        scopes:
          type: array
          items: { type: string }
        token_id:
          type: string
          description: Server-issued identifier for this token

    LongTokenRequest:
      type: object
      description: One of several credential mechanisms must be provided.
      properties:
        grant_type:
          type: string
          enum: [ client_credentials, assertion ]
        client_id:
          type: string
        client_secret:
          type: string
          description: Required if using client_credentials with secret
        otp_code:
          type: string
          description: Required if your policy mandates OTP for issuance
        assertion_type:
          type: string
          description: Required when grant_type=assertion
        assertion:
          type: string
          description: Signed JWT or other accepted assertion
        scopes:
          type: array
          items: { type: string }
          description: Scopes to assign to the long-term token
        ttl_seconds:
          type: integer
          minimum: 3600
          maximum: 7776000
          description: Requested lifetime (1 hour – 90 days). Server may clamp.
      required: [ grant_type, client_id ]
      additionalProperties: false

    LongTokenResponse:
      type: object
      required: [ token_type, access_token, expires_in, expires_at ]
      properties:
        token_type:
          type: string
```

```
          enum: [ Bearer ]
        access_token:
          type: string
          description: Long-lived token (opaque or JWT depending on deployment)
        expires_in:
          type: integer
        expires_at:
          type: string
          format: date-time
        scopes:
          type: array
          items: { type: string }
        token_id:
          type: string

    Error:
      type: object
      required: [ code, message ]
      properties:
        code:
          type: string
          example: invalid_grant
        message:
          type: string
          example: The provided client credentials are invalid.
        details:
          type: object
```

🧠 Design notes (sane defaults you can keep or tweak)

- **Short-term token**: 15 minutes (`expires_in: 900`). Minted from a **valid long-term token**. Supports **scope narrowing** (least privilege).
- **Long-term token**: 30–90 days by policy. Requires **step-up** (client secret, OTP, or signed assertion). Let the server clamp `ttl_seconds`.
- **Revocation**: include `token_id` in responses so clients can revoke later.
- **Security schemes**: I kept them simple (`http` bearer + optional `apiKey`), which works great in Postman.
- **Error model**: OAuth-ish `code` strings (e.g., `invalid_grant`, `invalid_scope`, `rate_limited`) make debugging easier.

🪛 Postman tips (flat collection)

- Import this spec with **Folder Organization = None** (or via API with `folderStrategy: "none"`) to keep the three requests flat.
- Add a **Pre-request Script** on your operational folders to **auto-exchange** a long token → short token when the short token is expired.
- Keep `{{long_token}}`, `{{short_token}}`, and `{{token_expires_at}}` as collection variables.

Example pre-request snippet (JS) to refresh short token on demand:

```
javascript

const expiresAt = pm.collectionVariables.get('token_expires_at');
if (!expiresAt || Date.now() > Number(expiresAt)) {
  pm.sendRequest({
    url: pm.environment.get('baseUrl') + '/auth/tokens/short',
```

```
    method: 'POST',
    header: {
      'Authorization': 'Bearer ' + pm.collectionVariables.get('long_token'),
      'Content-Type': 'application/json'
    },
    body: { mode: 'raw', raw: JSON.stringify({ scopes: pm.collectionVariables.get('scopes') ||
[] }) }
  }, (err, res) => {
    if (err) { console.log('Short token refresh failed', err); return; }
    const data = res.json();
    pm.collectionVariables.set('short_token', data.access_token);
    pm.collectionVariables.set('token_expires_at', String(Date.parse(data.expires_at)));
  });
}
```

Then set your request **Authorization** to `Bearer {{short_token}}`.

If you want, I can also:

- Generate a **standalone YAML file** you can download,
- Add **Spectral rules** to enforce token fields,
- Or wire a **Makefile target** to import this spec into Postman with `folderStrategy: none`.

> Currently the Makefile generates the openapi spec from a datadictionary ebnf - using a custom translator. Can you update the following ebnf to inlcude the data dictionary components necessary to generate the two jwt endpoints. Or would it be bette to generate the yaml for the jwt and add it to the openapi spec after the ebnf to openapi translation (using a Makefile target). Not sure what the spectral rules are for

Short answer: **don't push auth flows into your EBNF.** Keep the EBNF focused on your *business data dictionary* (documents, recipients, jobs…). Then **layer the JWT endpoints as a small OpenAPI overlay YAML** and merge it into the generated spec in your Makefile. This keeps concerns clean, avoids complicating your translator, and is easy to iterate.

Below I give you:

1. a tiny EBNF update (optional) to declare token *schemas* only,
2. a ready-to-paste **auth overlay YAML** with the two endpoints,
3. Makefile targets to **merge** the overlay into the generated spec, and
4. a **Spectral** snippet (what it is + minimal rules) to lint the result.

---

1) (Optional) Minimal EBNF additions — schemas only

If your translator already turns EBNF "record" definitions into OpenAPI `components/schemas`, you can add just the token models here (no endpoints). That lets you reuse these schemas in other places if needed.

```ebnf
(***********************************)
(*      AUTH / TOKEN DATA TYPES    *)
(***********************************)

ShortTokenRequest =
    [ scopes: { string } ]           (* optional list of scopes *)
  | [ longToken: string ] ;          (* optional if Authorization header carries it *)

ShortTokenResponse =
    tokenType: ("Bearer")
  + accessToken: string
  + expiresIn: integer               (* seconds, e.g., 900 *)
  + expiresAt: datetime              (* RFC3339 *)
  + [ scopes: { string } ]
  + [ tokenId: string ] ;

LongTokenRequest =
    grantType: ("client_credentials" | "assertion")
  + clientId: string
  + [ clientSecret: string ]         (* when using client_credentials *)
  + [ otpCode: string ]              (* if your policy requires step-up *)
  + [ assertionType: string ]        (* when grantType = assertion *)
  + [ assertion: string ]            (* signed JWT or similar *)
  + [ scopes: { string } ]
  + [ ttlSeconds: integer ] ;        (* 3600..7776000 *)

LongTokenResponse =
    tokenType: ("Bearer")
  + accessToken: string
  + expiresIn: integer
  + expiresAt: datetime
  + [ scopes: { string } ]
  + [ tokenId: string ] ;

AuthError =
    code: string
  + message: string
  + [ details: object ] ;
```

> Your translator will map these to **`components/schemas`**. **Do not** try to express verbs/paths in EBNF—leave endpoints to the overlay YAML below.

---

2) Auth overlay YAML (drop-in)

Save this as **`openapi/overlays/auth.tokens.yaml`**. It defines the two endpoints and reuses the schemas (either the translator-generated ones above **or** the inline ones here if you skip EBNF changes).

```yaml
openapi: 3.0.3
info:
  title: C2M API v2 — Auth Overlay
  version: 2.0.0

paths:
  /auth/tokens/short:
    post:
      tags: [auth]
      summary: Issue a short-term access token
      description: Exchange a valid long-term token for a short-lived JWT (e.g., 15 minutes).
      security:
        - LongTokenAuth: []
      requestBody:
        required: false
```

```yaml
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ShortTokenRequest'
      responses:
        '201':
          description: Short-term token issued
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ShortTokenResponse'
        '400': { $ref: '#/components/responses/Error400' }
        '401': { $ref: '#/components/responses/Error401' }
        '429': { $ref: '#/components/responses/Error429' }

  /auth/tokens/long:
    post:
      tags: [auth]
      summary: Issue or rotate a long-term token
      description: Issue a long-lived token (e.g., 30–90 days) via client credentials, OTP, or
signed assertion.
      security:
        - ShortTokenAuth: [tokens:write]
        - ClientKey: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/LongTokenRequest'
      responses:
        '201':
          description: Long-term token issued
          headers:
            X-Token-Id:
              description: Identifier for revocation/audit
              schema: { type: string }
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/LongTokenResponse'
        '400': { $ref: '#/components/responses/Error400' }
        '401': { $ref: '#/components/responses/Error401' }
        '403': { $ref: '#/components/responses/Error403' }
        '429': { $ref: '#/components/responses/Error429' }

  /auth/tokens/{tokenId}/revoke:
    post:
      tags: [auth]
      summary: Revoke a token
      description: Revokes a short- or long-term token by `tokenId`. Idempotent.
      security:
        - ShortTokenAuth: [tokens:revoke]
        - LongTokenAuth:  [tokens:revoke]
      parameters:
        - name: tokenId
          in: path
          required: true
          schema: { type: string }
      responses:
        '204':
          description: Revoked
        '401': { $ref: '#/components/responses/Error401' }
        '404': { $ref: '#/components/responses/Error404' }

components:
  securitySchemes:
    LongTokenAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
    ShortTokenAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

```yaml
    ClientKey:
      type: apiKey
      in: header
      name: X-Client-Id

  # If your EBNF->OpenAPI already emits these schemas, you can remove this block
  schemas:
    ShortTokenRequest:
      type: object
      properties:
        long_token: { type: string, description: Optional if sent via Authorization header }
        scopes: { type: array, items: { type: string } }
      additionalProperties: false

    ShortTokenResponse:
      type: object
      required: [token_type, access_token, expires_in, expires_at]
      properties:
        token_type: { type: string, enum: [Bearer] }
        access_token: { type: string }
        expires_in: { type: integer }
        expires_at: { type: string, format: date-time }
        scopes: { type: array, items: { type: string } }
        token_id: { type: string }

    LongTokenRequest:
      type: object
      required: [grant_type, client_id]
      properties:
        grant_type: { type: string, enum: [client_credentials, assertion] }
        client_id: { type: string }
        client_secret: { type: string }
        otp_code: { type: string }
        assertion_type: { type: string }
        assertion: { type: string }
        scopes: { type: array, items: { type: string } }
        ttl_seconds:
          type: integer
          minimum: 3600
          maximum: 7776000
      additionalProperties: false

    LongTokenResponse:
      type: object
      required: [token_type, access_token, expires_in, expires_at]
      properties:
        token_type: { type: string, enum: [Bearer] }
        access_token: { type: string }
        expires_in: { type: integer }
        expires_at: { type: string, format: date-time }
        scopes: { type: array, items: { type: string } }
        token_id: { type: string }

    Error:
      type: object
      required: [code, message]
      properties:
        code: { type: string }
        message: { type: string }
        details: { type: object }

  responses:
    Error400:
      description: Bad request
      content:
        application/json:
          schema: { $ref: '#/components/schemas/Error' }
    Error401:
      description: Unauthorized
      content:
        application/json:
          schema: { $ref: '#/components/schemas/Error' }
    Error403:
      description: Forbidden
      content:
```

```
        application/json:
          schema: { $ref: '#/components/schemas/Error' }
    Error404:
      description: Not found
      content:
        application/json:
          schema: { $ref: '#/components/schemas/Error' }
    Error429:
      description: Too Many Requests
      content:
        application/json:
          schema: { $ref: '#/components/schemas/Error' }
```

---

3) Makefile: merge overlay into the generated spec

Assume your translator outputs **`openapi/build/openapi.generated.yaml`**. We'll merge overlays (auth now, others later) using **`yq`** (go-yq v4).

```
make


# --- Files/dirs you likely already have ---
OPENAPI_GEN := openapi/build/openapi.generated.yaml
OPENAPI_FINAL := openapi/build/openapi.final.yaml

# --- Auth overlay ---
OPENAPI_AUTH_OVERLAY := openapi/overlays/auth.tokens.yaml

# Merge function (base *+ overlay = deep merge, overlay wins)
.PHONY: openapi-merge-auth
openapi-merge-auth: $(OPENAPI_GEN) $(OPENAPI_AUTH_OVERLAY)
        @echo "🔗 Merging auth overlay into generated OpenAPI..."
        @yq ea -o=yaml 'select(fileIndex == 0) *+ select(fileIndex == 1)' \
                $(OPENAPI_GEN) $(OPENAPI_AUTH_OVERLAY) > $(OPENAPI_FINAL)
        @echo "✅ Wrote $(OPENAPI_FINAL)"

# Full build: generate from EBNF, then merge auth, then lint, then (optionally) import to
Postman
.PHONY: openapi-build-all
openapi-build-all: ebnf-to-openapi openapi-merge-auth spectral-lint  ## end-to-end
        @echo "🎉 OpenAPI built + auth merged + linted: $(OPENAPI_FINAL)"
```

If you maintain multiple overlays, extend the **`yq ea`** line: **`base *+ overlay1 *+ overlay2 *+ …`**

Optional: import the final spec into Postman **flattened** (no folders):

```
make


POSTMAN_BASE_URL ?= https://api.getpostman.com
POSTMAN_API_KEY  ?= $(POSTMAN_API_KEY)
POSTMAN_WS       ?= <your_workspace_id>
POSTMAN_LAST_IMPORT_JSON := .tmp/postman_last_import.json

.PHONY: postman-import-openapi-flat
postman-import-openapi-flat: $(OPENAPI_FINAL)
        @mkdir -p .tmp
        @echo "📥 Importing $(OPENAPI_FINAL) to Postman (flat)..."
        @curl --silent --location --request POST "$(POSTMAN_BASE_URL)/import/openapi?
workspace=$(POSTMAN_WS)" \
                -H "X-Api-Key: $(POSTMAN_API_KEY)" \
                --form "file=@$(OPENAPI_FINAL)" \
                --form 'parameters={"folderStrategy":"none"}' \
        | tee $(POSTMAN_LAST_IMPORT_JSON) >/dev/null
        @echo "✅ Imported. Collection UID:" $$(jq -r '.collections[0].uid // .collection.uid'
$(POSTMAN_LAST_IMPORT_JSON))
```

4) What Spectral is (and minimal rules you might want)

Spectral is a linter for OpenAPI/JSON/YAML. You add a ruleset and run it in CI to catch issues like "missing 4xx responses", "undocumented security", "non-RFC date-time", etc.

Create `.spectral.yaml`:

```
yaml

extends: ["spectral:recommended", "spectral:asyncapi", "spectral:oas"]
formats: ["oas3"]
rules:
  # Ensure auth endpoints return 201
  auth-created-201:
    description: Auth token creation endpoints should return 201
    given: $.paths[?(@property.match(/\/auth\/tokens\/(short|long)$/))].post.responses
    then:
      field: "201"
      function: truthy

  # Require bearer security on /auth/*
  auth-requires-security:
    description: /auth/* endpoints must declare security
    given: $.paths[?(@property.match(/^\/auth\//))].*
    then:
      field: security
      function: truthy

  # Token response must include token_type/access_token/expires_in
  token-response-shape:
    description: Token responses should include token_type, access_token, expires_in
    given: $.components.schemas[*]
    then:
      function: schema
      functionOptions:
        schema:
          type: object
          properties:
            token_type: { type: string }
            access_token: { type: string }
            expires_in: { type: integer }
          required: [token_type, access_token, expires_in]
          additionalProperties: true
    severity: warn
```

Makefile target:

```
make

.PHONY: spectral-lint
spectral-lint: $(OPENAPI_FINAL)
        @npx -y @stoplight/spectral-cli lint $(OPENAPI_FINAL)
```

TL;DR recommendation

- **Preferred**: keep EBNF for business data; **merge auth endpoints via an overlay YAML** (simple Makefile target above).
- **Optional**: declare token *schemas* in EBNF if you want them produced by your translator.
- Use **Spectral** to keep everyone honest (fields present, security declared, correct codes).

- Import to Postman with `folderStrategy: "none"` to keep the new endpoints flat alongside your other calls.

If you drop me the names/paths of your current Makefile targets (`ebnf-to-openapi`, output locations, etc.), I'll align the variable names and wire this straight into your pipeline.