

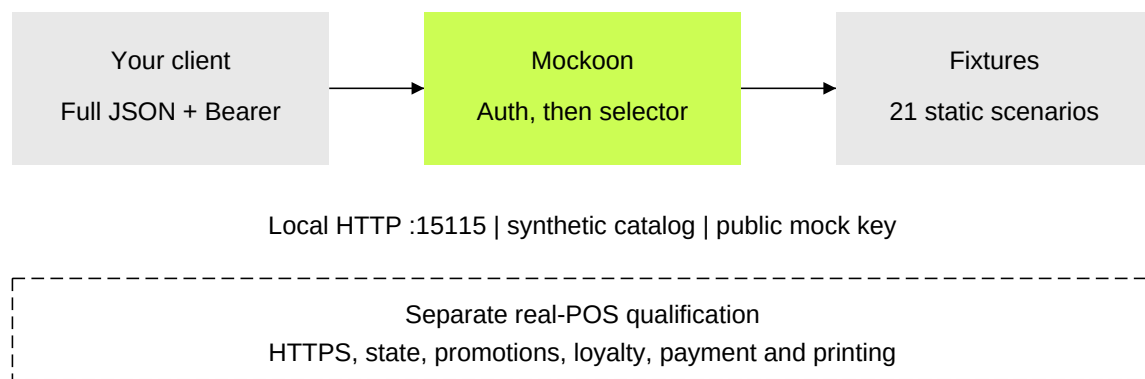
Touchpoint integration guide

Version 0.2.0 | 15 October 2026 | Standard czz-src integration kit

1. What you can integrate

Use this kit to develop a Touchpoint HTTP client without a running POS. Mockoon serves complete, synthetic responses for basket calculation, confirmation, retries and failures. All responses are **SIMULATED**, derived from a source contract, not captured from a real till. CDS is one consumer of this standard API; its deployment and catalog are not prerequisites.

The environment has 21 explicit scenario endpoints and two real-path aliases. It checks a public mock credential and lane before selecting a response. It does not calculate prices, query loyalty, park a basket or maintain state. A successful smoke test proves the packaged HTTP fixtures; it does not qualify production security, persistence, idempotency, concurrency, payment or printing.



Extract the complete ZIP, then run commands from its top-level directory. You need either Docker with Compose, Mockoon Desktop, or Node.js 22.23.2 and the pinned Mockoon CLI 9.0.0. Initial Docker pulls and npm installation require internet access. No private repository access or real credentials are needed.

The hosted portal is <https://api.pos.lab.retexcloud.io>. Use Swagger there without installation, or select Download ZIP and SHA-256 checksum to obtain the same version for local use. The download includes this guide, OpenAPI, Mockoon fixtures, Swagger assets and Compose. No access to the private repository is needed. Send synthetic inputs only to the public simulation.

Verify the downloaded archive before extraction:

```
shasum -a 256 -c czz-touchpoint-integration-kit-0.2.0.zip.sha256
```

Use the version shown in the downloaded filename. After extraction, follow the local startup below. The public portal and local portal use the same scenarios; /mock is the portal proxy prefix on both.

2. Start with Docker Compose

The supplied [compose.yaml](#) starts Mockoon CLI 9.0.0 and the kit web server. It mounts the package read-only and publishes the API on port 15115 and the portal on port 15116, both on loopback only. Swagger UI assets are bundled in the ZIP. No Node.js or Python installation is needed on the host when using Compose.

```
docker compose -p czz-touchpoint-kit up -d
docker compose -p czz-touchpoint-kit logs --tail 20
```

Open <http://127.0.0.1:15116>. Select Run the first item, then press Execute in Swagger UI. The public mock key, request body and X-Mock-Case: 02 selector are already filled in. Expect HTTP 200, one accepted item, total 2 and saving 1. The response is displayed under Server response.

The portal links all 21 cases and the sequence 02 → 03 → 04 → 14 → 10 → 10r (items, loyalty, finish, retry). Each preset opens the corresponding calculate/finish alias with a matching body and selector. Use the Integration alias preset dropdown to change both together. Changing only Swagger's request example does not change the header: keep them consistent.

Open <http://127.0.0.1:15116/swagger/> directly for the full endpoint catalog. Download [openapi.json](#) for your own tooling. The bundled contract uses /mock as its server: this is the portal proxy prefix, not part of the production API path. Direct clients use port 15115 without /mock.

To test denied authentication, open Authorize, choose Logout, close the dialog and execute again: expect HTTP 403. Reload the page to restore the public demo key. To test selector validation, choose MOCK_ONLY_INVALID_SELECTOR for X-Mock-Case: expect HTTP 400. Case 09 covers stale confirmation, case 10c conflict after payment, and case 14unavailable a loyalty failure.

You can also make the same first request from a terminal:

```
curl --fail-with-body -sS \
  http://127.0.0.1:15115/caso02 \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer MOCK_ONLY_STORE_A_KEY_NOT_A_REAL_CREDENTIAL' \
  --data-binary @cases/02/request.json
```

Expected: HTTP 200, basketId = mock-basket-001, one accepted line a1, total = 2 and totalSaving = 1. The complete expected body is [case 02 response](#).

```
docker compose -p czz-touchpoint-kit down
```

If either port is occupied, use MOCK_PORT=15125 PORTAL_PORT=15126 `docker compose -p czz-touchpoint-kit-alt up -d`. Open <http://127.0.0.1:15126> and use port 15125 for direct API calls. The browser proxy continues to work without changes. Stop this separate instance with `docker compose -p czz-touchpoint-kit-alt down`. Do not run Desktop, CLI and Docker on the same host port. Container shutdown removes no POS data; this kit has no database.

3. Other startup methods

Docker without Compose

```
docker run --rm --name czz-touchpoint-kit \
  -p 127.0.0.1:15115:15115 \
  -v "$PWD/environment.json:/data/environment.json:ro" \
  mockoon/cli:9.0.0 \
  --data /data/environment.json --hostname 0.0.0.0 --port 15115
```

Test from another terminal. Stop with Ctrl-C. The host API is `http://127.0.0.1:15115`; clients in the Compose network use `http://mockoon:15115`. The container listens internally on all interfaces, but the host mapping stays on loopback. Keep this public-credential mock local.

Mockoon Desktop

In Mockoon Desktop 9.0.0, use Open local data file to open `environment.json`. Select czz Touchpoint integration kit 0.2.0 - SIMULATED, retain hostname `127.0.0.1` and port `15115`, then start the environment. Test the first request before continuing. Stop it before another runtime uses that port. Use a copy if a newer app proposes migration. Desktop interaction remains unverified; CLI import of the same file is verified.

CLI and automated tests

Use Node.js 22.23.2. The lockfile pins Mockoon CLI 9.0.0 and its dependencies.

```
npm ci --ignore-scripts --no-audit --no-fund
npm start
```

Keep that terminal open. In a second terminal at the package root:

```
npm test
```

`npm test` requires a running server at `http://127.0.0.1:15115`. For another local port:

```
MOCK_BASE_URL=http://127.0.0.1:15116 npm test
```

Tests deliberately reject non-loopback targets. Stop the CLI with Ctrl-C. The same test command can qualify the Compose or direct Docker instance.

Swagger UI with Desktop, CLI or standalone Docker

Start Mockoon using one of the methods above. In another terminal at the extracted package root, run the bundled web server with Python 3.10 or later:

```
python3 scripts/serve.py
```

Open `http://127.0.0.1:15116`. No pip packages or npm installation are required for this server. It serves the bundled UI and forwards browser requests to Mockoon on port `15115`. For custom ports:

```
MOCK_BASE_URL=http://127.0.0.1:15125 python3 scripts/serve.py --port 15126
```

Stop the web server with Ctrl-C. Do not open `web/swagger.html` directly as a file: the web server supplies the contract, examples and same-origin proxy. A 502 response means the configured mock backend is not reachable; start Mockoon and check the selected port. If the portal does not open, inspect Compose logs or the Python terminal and check for a port conflict.

4. Authentication from the client

For a real installation, obtain the store API key, POS HTTPS endpoint and assigned touchpointId from the installation contact. Send the API key on every request as `Authorization: Bearer <api-key>`, with `Content-Type: application/json`. The lane assignment must match the destination POS. Bearer is the credential transport: this contract has no JWT generation, decoding, renewal, OAuth or login endpoint.

For this mock only, use the public string `MOCK_ONLY_STORE_A_KEY_NOT_A_REAL_CREDENTIAL` and lane `SCO-01`. This string is not a real credential. A wrong or absent key, wrong scheme or unauthorized lane returns HTTP 403 with exactly:

```
{"error":"Touchpoint authorization denied"}
```

Do not put real keys in URLs, request JSON or logs. Store them using the client platform's credential facilities; verify the HTTPS server certificate. On the documented authorization rejection, stop repetitive retries and request verification of key, destination and lane assignment. Other 403 bodies can represent application failures, such as case 18. Do not automatically request a new JWT.

The [authentication fixtures](#) provide separate request, headers and response files for valid, missing, wrong-scheme, wrong-key, other-store, revoked and wrong-lane cases. Other-store and revoked keys are static example strings, not a credential issuance or revocation service. The mock applies this check before X-Mock-Case; selecting a positive case cannot bypass it.

```
curl -i http://127.0.0.1:15115/caso02 \
-H 'Content-Type: application/json' \
--data-binary @cases/auth/missing/request.json
```

The authorization contract references draft implementation #3334 at revision `ce2e19d649251e59e47c2d4a16dae06f13b7eaf6`; this is candidate evidence, not a released remote listener. Confirm actual deployment support before a POS integration. A future mTLS mode requires separate implementation and qualification. It is out of scope and is not an available kit option. This kit covers HTTP only and provides no WebSocket mock.

5. Endpoint selection

All scenario endpoints use POST. Send the complete matching request file and the headers in its headers . json. Static responses do not validate arbitrary basket content; use the scenario fixtures as full inputs. Authorization still checks the submitted lane.

The endpoint catalog below lists every scenario with links to request and response. The machine-readable [index](#) also records method, status, real-path alias, selector, predecessors and provenance.

POST endpoint	Scenario	HTTP	Files
/caso01	Empty basket; dedicated operator available	200	request / response / headers
/caso02	First item	200	request / response / headers
/caso03	Second item, two units	200	request / response / headers
/caso04	Third item	200	request / response / headers
/caso06	Remove item a2	200	request / response / headers
/caso07	Unknown item	200	request / response / headers
/caso08	Duplicate reference	200	request / response / headers
/caso09	Stale digest; ask for confirmation again	200	request / response / headers
/caso10	Finish accepted	200	request / response / headers
/caso10r	Repeat finish	200	request / response / headers
/caso10c	Finish after payment	409	request / response / headers
/caso10claimed	Finish after manual retrieval	409	request / response / headers
/caso14	Loyalty accepted	200	request / response / headers
/caso14refused	Loyalty refused	200	request / response / headers
/caso14unavailable	Loyalty service unavailable	200	request / response / headers

POST endpoint	Scenario	HTTP	Files
/caso15	Card sent as an item	200	request / response / headers
/caso16	Price-embedded label	200	request / response / headers
/caso16b	Label quantity rejected	200	request / response / headers
/caso17	Age check enabled; item refused	200	request / response / headers
/caso18	Touchpoints disabled	403	request / response / headers
/caso19	Dedicated operator unavailable	503	request / response / headers

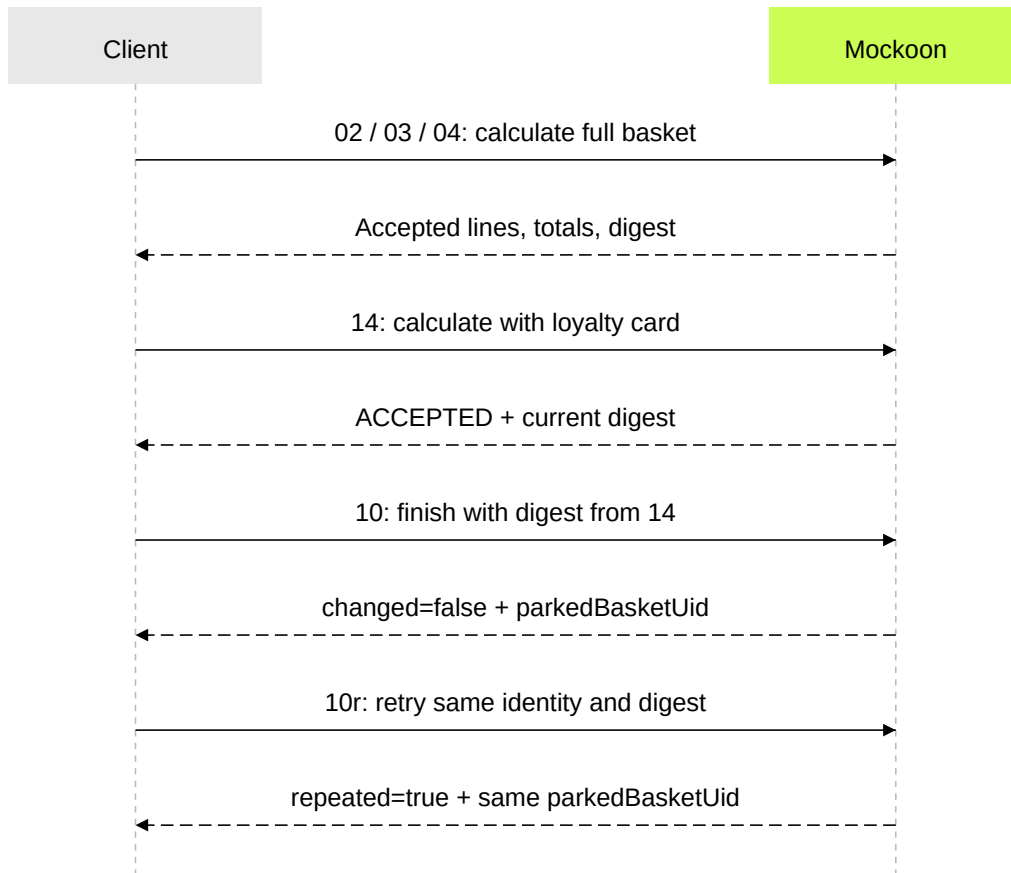
For the client paths, use one of these aliases:

- /comerzzia/touchpoint/basket/calculate: 01, 02, 03, 04, 06, 07, 08, 14, 14refused, 14unavailable, 15, 16, 16b, 17, 18, 19.
- /comerzzia/touchpoint/basket/finish: 09, 10, 10r, 10c, 10claimed.

Add X-Mock-Case: <case-id> to select the response. This header is mock-only; remove it when connecting to a real POS. With valid authorization, an absent, unknown or incompatible selector returns HTTP 400 and {"error":"MOCK_ONLY: missing, unknown or incompatible X-Mock-Case selector"}. There is no implicit positive default.

```
curl --fail-with-body -sS \
  http://127.0.0.1:15115/comerzzia/touchpoint/basket/finish \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer MOCK_ONLY_STORE_A_KEY_NOT_A_REAL_CREDENTIAL' \
  -H 'X-Mock-Case: 10' \
  --data-binary @cases/10/request.json
```

6. Follow the basket sequence



Static selected responses; no state or idempotency proof.

The caller owns the current basket input. Carry accepted line identity and the server's digest forward. A returned digest is opaque to the client; do not recalculate it. Case files are complete snapshots so each can also be called independently. These sequences describe client behavior; the static mock neither enforces order nor remembers requests.

Path	Ordered steps	Data and client checks
Build	01 -> 02 -> 03 -> 04	Keep basketId; carry accepted lines, add a1, then a2 qty 2, then a3. Totals: 0, 2, 6, 9.5.
Remove	04 -> 06	Remove a2; retain a1 and a3. Show total 5.5.
Reject	04 -> 07 or 08	Independent branches. Preserve accepted lines; show x1 NOT_FOUND or a1 DUPLICATE_REF. Do not add rejected input.
Stale digest	04 -> 09	Finish uses a deliberately stale digest. changed=true and parkedBasketUid=null: display the updated basket and require confirmation again.

Path	Ordered steps	Data and client checks
Loyalty and finish	04 -> 14 -> 10 -> 10r	Add loyaltyCard. Copy digest from 14 into finish 10; retain the same identity and digest for retry 10r. Show repeated=true and the same UID.
Claimed	10 -> manual retrieval -> 10claimed	Explicit simulated 409; stop finish retries and follow operator recovery.
Paid	10 -> manual retrieval/payment -> 10c	Explicit simulated 409; do not resubmit this closed basket as a new sale.
Loyalty alternatives	04 -> 14refused or 14unavailable	Independent alternatives: explain REFUSED or UNAVAILABLE to the user. Do not treat either as ACCEPTED.
Special barcode	02 -> 15, 16, 16b or 17	Independent branches: card-as-line rejected, weighed label accepted, label quantity rejected, age-check refusal.
Configuration	18 or 19	Simulate disabled endpoint or missing operator; require installation support. No real configuration changes occur.

After case 09, a real client uses the returned basket and digest for a fresh user confirmation. The kit has no accepted finish fixture for that no-card basket. Case 10 belongs to the separate 14 -> 10 path and must not be described as a direct continuation of 09. Restart any mock path by loading its initial full request; no reset API or server restart is required. Calling 10r without 10 still returns the static retry fixture and proves no idempotency.

Each step's request and response are linked in the catalog. The previous arrays in the index list predecessor cases, while the table above specifies branches and manual boundaries.

Manual POS work

UAT case	Procedure outside Mockoon
5	Compare the displayed basket with the real till. No HTTP request/response applies.
11	Retrieve the parked basket using F1 on the actual POS. No fabricated endpoint.
12	Complete payment and verify printed output on actual hardware.
13	Run 02 -> 03 -> 04 and manually verify that the cashier's current basket has not changed.
20	Reuse finish 10 as the mock example; inspect the operator of the real parked basket separately.

7. Data model and synthetic catalog

Field	Client responsibility
touchpointId	Send the assigned lane; SCO-01 in this mock.
basketId	Keep one stable identity throughout calculation and finish/retry.
lines	Send the complete retained basket input, with ref, barcode and qty.
add	Optional single new scan; do not duplicate an already retained ref.
ref	Stable client line identity; use a new value for a distinct insertion.
loyaltyCard	Send the card separately from article lines. Only synthetic identifiers are included.
loyaltyCardStatus	Server outcome: ACCEPTED, REFUSED or UNAVAILABLE; null when no card applies.
digest	Opaque server fingerprint. Retain the latest returned value for finish.
rejected	Per-line refusal even when HTTP is 200; inspect code and ref.
changed	On finish, true requires renewed user confirmation; it is not successful parking.
repeated	Marks the simulated repeated finish result. Preserve the original operation identity.
parkedBasketUid	Server-issued identity for the accepted parked basket; null on stale digest.
parkedAlias	Display reference; SCO-01 in accepted finish fixtures.

Prices, discounts, totals, item descriptions, lineId and promotions are server output. Do not derive authoritative totals from client scan input. The kit's engine=MockFixture is an explicit simulation label, not a production engine name.

Item	Quantity	Unit price	Amount	Saving
a1 / MOCK-ITEM-A	1	3.00	2.00	1.00
a2 / MOCK-ITEM-B	2	2.50	4.00	1.00
a3 / MOCK-ITEM-C	1	4.00	3.50	0.50
w1 / 2000653002503	One label, 0.2 kg	12.50 per kg	2.50	0.00

Case 16 assumes the example label encodes amount 2.50 and resolves to synthetic DEMO-W. No barcode mask is actually decoded. Quantity must be one scan per label in this scenario; case 16b rejects qty 3. The age-check feature in case 17 is a declared simulated precondition, not a universal behavior of all installations. Promotion identifiers 90001-90003 are synthetic. See [catalog.json](#).

8. Troubleshooting

Symptom	Check and action
Connection refused	Start exactly one runtime; wait for its listening log. Verify host port and host/container address.
Port already allocated	Stop the other mock or change the host mapping and client base URL together.
Environment missing in Docker	Run from the extracted root; verify environment.json exists and the bind mount points to a file, not a directory.
Import or migration failure	Use version 9.0.0 with the supplied unmodified environment; preserve an original copy before newer Desktop migration.
Generic authorization 403	Send Bearer plus the complete public mock key and touchpointId SCO-01. Check whitespace and header spelling.
MOCK_ONLY 400	On aliases, provide an allowed X-Mock-Case for that operation. Explicit /caso routes need no selector.
HTTP 200 but scan absent	Inspect rejected. A line refusal is not necessarily an HTTP error.
Response ignores changed prices	Expected: fixtures are static. Modify the generator and regenerate fixtures for a different simulated contract.
Finish retry succeeds without earlier call	Expected: 10r is selected explicitly; no transaction state exists.
Node engine warning	Use Node 22.23.2, then reinstall with npm ci.

9. Integration checklist and real-POS transition

- Parse every accepted line and rejected line, not only HTTP status.
- Keep basket identity, refs and current digest across calls; test stale-digest confirmation separately from retry.
- Handle REFUSED and UNAVAILABLE loyalty outcomes explicitly.
- Check 403 authorization, 403 disabled, 409 claimed/closed and 503 operator errors separately.
- Remove X-Mock-Case and use the actual POS HTTPS endpoint and installation-issued store key.
- Confirm the deployed authentication/listener contract and lane assignment with the installation contact.
- Qualify real catalog, promotions, label masks, optional age checks and loyalty availability.
- Test persistence, concurrent clients, true repeated finish, recovery, payment and physical printing on a real POS. No mock result attests those behaviors.

10. Provenance, regeneration and verification

The contract and authentication revisions are recorded in [provenance.json](#). Every scenario has SIMULATED provenance. There are zero captured responses. Customer-specific mappings are isolated in [examples/cds](#); no real customer identifiers are distributed.

The document is an A4 technical adaptation of [nomed/alkemy-art](#) revision `6e0013c8e7e6941b54c9c404a90277ab873cac04`, using its canonical palette, Arial profile and original logo. It is not an existing A4 template from that repository. Diagram Design 2.6 sequence/layout guidance is applied with these Alkemy tokens; semantic Mermaid sources and editable SVG/HTML are included. See [third-party notices](#).

Use the commands in the [maintainer README](#) to regenerate fixtures, document tables, PDF and the deterministic ZIP. PDF and Markdown use the same guide content. The ZIP contains all runtime input and documentation; PDF regeneration additionally requires the pinned Python dependencies.

CLI 9.0.0 / Node 22.23.2 and Docker Compose with `mockoon/cli:9.0.0` each passed 229 HTTP comparisons on 15 October 2026. The tests include explicit scenarios, aliases, auth failures and selector errors. Direct Docker and extracted-package verification are recorded in `VALIDATION.md`. Desktop interaction and real-POS behavior are not qualified.

Appendix: complete example input and output

The files linked in the endpoint catalog contain the complete request, response and headers for every case. The following first-scan example is also embedded for offline reading.

Request:

```
{
  "touchpointId": "SCO-01",
  "basketId": "mock-basket-001",
  "lines": [],
  "add": {
    "ref": "a1",
    "barcode": "MOCK-ITEM-A",
    "qty": 1
  }
}
```

Response (HTTP 200):

```
{
  "touchpointId": "SCO-01",
  "basketId": "mock-basket-001",
  "engine": "MockFixture",
  "loyaltyCard": null,
  "loyaltyCardStatus": null,
  "lines": [
    {
      "ref": "a1",
      "barcode": "MOCK-ITEM-A",
      "qty": 1,
      "lineId": 1,
      "itemCode": "DEMO-A",
      "description": "Demo item A",
      "weight": null,
      "unitPrice": 3,
      "amountWithoutDiscount": 3,
      "amount": 2,
      "discount": 1,
      "promotions": [
        {
          "promotionId": 90001,
          "text": "Synthetic demonstration discount",
          "saving": 1
        }
      ]
    }
  ],
  "rejected": [],
  "totalItems": 1,
  "total": 2,
  "totalSaving": 1,
  "changed": null,
  "repeated": null,
  "parkedBasketUid": null,
  "parkedAlias": null,
  "digest": "0673ac4c766a437914848495ce9dbb1ce543f124a6cfbee43b838fe0b2f26de4"
}
```

PDF export font: Liberation Sans (Arial-compatible). The canonical Alkemy profile is Arial; the release pipeline uses the declared Arial-compatible Liberation Sans fallback.