SPEC-1 – Malaysia QTC Simulation Dashboard

Background

Purpose. Build an interactive simulation and monitoring dashboard that lets Malaysian policymakers test how *steering the composition and quantity of bank credit* (per Richard Werner's Quantity Theory of Credit) affects real activity, inflation, asset prices, and financial stability.

Economic basis.

- Banks create money individually via lending; outcomes depend on *what credit is created for*.
- Disaggregate credit into **productive/GDP** vs **non-GDP/asset** uses; composition drives growth vs bubbles.
- Direct/quantitative credit guidance (by loan class/sector) is the primary policy lever; interest rates are secondary.

Malaysia context (motivation).

- Bank Negara Malaysia (BNM) already uses macroprudential tools (e.g., LTV/DTI, sectoral exposure limits) and publishes granular credit stats. A QTC-centric simulator would unify these into a forward-looking policy workbench.

What the simulator should capture (at a high level).

- Bank balance-sheet expansion mechanics and reclassification of loans into GDP vs non-GDP uses.
- Policy levers: sectoral credit quotas/paths, LTV/DTI/DSR caps, capital risk-weights, liquidity/FX limits, and (secondary) policy rate.
- Observable outcomes: real GDP growth, CPI, PPI, property & equity price indices, NPL/PD migration, external balance/FX sensitivity.
- Feedback loops: asset-price-credit amplification, credit misallocation → NPLs, and macroprudential tightening/loosening.
- Shocks: commodity price swings, export demand, exchange-rate moves, and capital-flow volatility.

Intended users (initial). BNM (Monetary Policy, Financial Stability, Supervision), MoF/EPU analysts, and optionally participating banks.

Decision cadence. Monthly/quarterly scenarios with 1–12 quarter horizons; daily feeds only for monitoring (not core to v1).

Requirements

Must-have (M)

- **R-M01 | Disaggregated credit model** — Track **monthly** credit stocks/flows by **purpose** (Productive/GDP vs Non-GDP/asset) and by **sector bucket** (Manufacturing, Agriculture, Services, Construction/Real Estate, SME, Household: Mortgages/Consumer, Corporate: Securities/CRE).
- **R-M02 | Policy levers** — Scenario editor for **quantitative credit guidance** (sector/purpose growth paths & caps), **macroprudential** (LTV/DTI/DSR, sector exposure caps), **capital/risk-weight tweaks**, **FX/foreign-borrowing limits**, and **policy-rate path** (secondary).
- **R-M03 | Exogenous shocks** — Toggleable shocks: property price, export demand, commodity terms-of-trade, FX.
- **R-M04 | Outputs** — Time series & charts for **Real GDP growth**, **CPI**, **Property index**, **Equity index**, **NPL ratio/PD migration**, **Credit composition shares**, **External balance proxy**.
- **R-M05 | Data ingestion** — Connectors for BNM statistical tables (credit by purpose/sector), DOSM GDP/CPI, MHPI (house prices), Bursa indices; CSV upload for custom series.
- **R-M06 | Reproducibility & versioning** — Deterministic runs (seeded), **scenario version control**, parameter snapshots, and audit trail of inputs/outputs.
- **R-M07 | Security & compliance** — RBAC, SSO (IdP integration), full **audit logs**, PDPA compliance, encryption in transit/at rest; **data residency** within BNM/ Malaysia.
- **R-M08 | Performance** — Single scenario (12-quarter horizon) $\leq$ **10 s** on standard analyst hardware; compare-two scenarios $\leq$ **20 s**.
- **R-M09 | Usability** — Web dashboard with scenario builder, side-by-side comparisons, export **CSV/PNG/PDF**.

Should-have (S)

- **R-S01 | Bank-group granularity (toggle)** — Optional top-bank clusters vs system-level.
- **R-S02 | Calibration tools** — Parameter estimation/back-testing vs history with goodness-of-fit metrics.
- **R-S03 | API access** — Read-only REST/GraphQL for scenarios & results; CLI client.
- **R-S04 | Monitoring mode** — Live dashboard of latest data, with rules-based alerts (e.g., non-GDP credit share > threshold).

Could-have (C)

- **R-C01 | Agent-based lending heuristics** for banks & borrowers.
- **R-C02 | ML nowcasting** for high-frequency GDP/CPI proxies.
- **R-C03 | Collaborative notebooks** (policy notes attached to scenarios).

Won’t-have in v1 (W)

- **R-W01 | Borrower-level microdata** ingestion (confidential/sensitive).
- **R-W02 | Real-time streaming** bank-by-bank exposures.
- **R-W03 | Automated supervisory actions** (recommend only; no write-back to core systems).

Acceptance criteria (MVP)

- **A-01** Reproduces sign & order-of-magnitude of Malaysia's key cycles (GDP vs property & equity) in back-tests.
- **A-02** Policy toggles change outcomes in expected directions (e.g., tightening LTV reduces house-price inflation and raises NPLs with a lag within modeled bands).
- **A-03** All runs are reproducible from stored scenarios; audit shows who, when, what.
- **A-04** Passes security review for internal deployment (RBAC, encryption, logs).

Method

1) Architecture Overview

We adopt a **modular monolith** for v1 (faster, auditable), deployable on **BNM on-prem/private cloud**. Modules communicate via in-process events; we keep clear seams to split into services later (Model Engine can be isolated first).

```
@startuml
skinparam componentStyle rectangle
skinparam shadowing false

package "QTC Simulator (v1)" {
  [Web UI (React 19)] as UI
  [API Gateway (FastAPI)] as API
  [AuthN/Z (Keycloak/OIDC)] as KC
  [Scenario Svc] as SC
  [Policy & Rules Svc] as PR
  [Model Engine] as ME
  [Data Ingestion] as DI
  [Time-series DB (Postgres+Timescale)] as TS
  [Obj Store (artifacts)] as OS
  [Audit/Logs] as AUD
}

UI --> API : HTTPS (JWT/OIDC)
API --> KC : OIDC
API --> SC : REST/JSON
API --> PR : REST/JSON
API --> ME : REST/JSON (async run)
SC --> TS : SQL/JSONB
PR --> TS : read rules
ME --> TS : read inputs / write outputs
DI --> TS : batch loads (BNM/DOSM/NAPIC/Bursa)
ME --> OS : save run artifacts
API --> AUD : events
@enduml
```

Why modular monolith? Single deployable with strict module boundaries simplifies **auditing, reproducibility, and security**; aligns with internal hosting/air-gap controls. We expose **read-only API** for results in v1.

---

2) Data Model & Schemas (PostgreSQL 17 + TimescaleDB hypertables)

Core enums

- `purpose_enum`: `PRODUCTIVE_GDP`, `NON_GDP_ASSET`.
- `sector_enum`: `AGRI`, `MANUF`, `SERV`, `CONST_RE`, `SME`, `HH_MORT`, `HH_CONS`, `CORP_SEC_CRE`.
- `freq_enum`: `D`, `W`, `M`, `Q`, `A`.
- `metric_enum`: `GDP_REAL_GROWTH`, `CPI_INFL`, `HOUSE_PRICE_INFL`, `EQUITY_RET`, `NPL_RATIO`, `CRED_PROD_GROWTH`, `CRED_ASSET_GROWTH`, `CAD_PCT_GDP`.

Tables (types abbreviated; PK underlined; `ts` is Timescale hypertable on `date`)

- `series_catalog` — registry of external series

- `` (uuid), `source` (text), `code` (text), `name` (text), `unit` (text), `freq` (freq_enum), `sector` (sector_enum? nullable), `purpose` (purpose_enum? nullable), `meta` (jsonb), `active` (bool)

- `series_data` (ts) — raw values & revisions

- `,` (date), `value` (numeric), `rev_id` (bigint), `ingested_at` (timestamptz), `checksum` (text)

- index: (`series_id`, `date`), compression policy, retention per source

- `credit_flow` (ts) — derived monthly flows by sector & purpose

- `,`, ``, `flow_amt` (numeric), `stock_amt` (numeric)

- `scenarios`

- `` (uuid), `name` (text), `created_by` (text), `created_at`, `notes` (text)

- `policy_paths` — quantitative guidance & macroprudential settings

- `,`, `sector` (sector_enum), `purpose` (purpose_enum), `credit_cap_growth_pct` (numeric), `ltv_cap` (numeric), `dti_cap` (numeric), `risk_weight_adj_bps` (int), `policy_rate` (numeric), `fx_borrowing_cap` (numeric), `flags` (jsonb)

- `shocks`

- `,` , `shock_type` (text), `magnitude` (numeric), `details` (jsonb)
- `runs`
- `` (uuid), `scenario_id`, `seed` (int), `status` (text), `started_at`, `finished_at`, `engine_version` (text), `params` (jsonb), `hash` (text)
- `outputs` (ts) — model results
- `,` , `metric` (metric_enum), `value` (numeric), `band_lo` (numeric), `band_hi` (numeric), `meta` (jsonb)
- `audit_log`
- `` (uuid), `who` (text), `what` (text), `when` (timestamptz), `details` (jsonb)

```
@startuml
skinparam shadowing false
class series_catalog {
  +series_id: uuid <<PK>>
  source: text
  code: text
  name: text
  unit: text
  freq: freq_enum
  sector: sector_enum
  purpose: purpose_enum
  meta: jsonb
  active: bool
}
class series_data {
  +series_id: uuid <<PK>>
  +date: date <<PK>>
  value: numeric
  rev_id: bigint
  ingested_at: timestamptz
  checksum: text
}
class credit_flow {
  +date: date <<PK>>
  +sector: sector_enum <<PK>>
  +purpose: purpose_enum <<PK>>
  flow_amt: numeric
  stock_amt: numeric
}
class scenarios { +scenario_id: uuid <<PK>>; name; created_by; created_at;
```

```
notes }
class policy_paths { +scenario_id <<PK>>; +date <<PK>>; sector; purpose;
credit_cap_growth_pct; ltv_cap; dti_cap; risk_weight_adj_bps; policy_rate;
fx_borrowing_cap; flags }
class shocks { +scenario_id <<PK>>; +date <<PK>>; shock_type; magnitude;
details }
class runs { +run_id: uuid <<PK>>; scenario_id; seed; status; started_at;
finished_at; engine_version; params; hash }
class outputs { +run_id <<PK>>; +date <<PK>>; +metric <<PK>>; value; band_lo;
band_hi; meta }
class audit_log { +event_id: uuid <<PK>>; who; what; when; details }

series_catalog ||--o{ series_data
credit_flow }o--|| series_catalog
scenarios ||--o{ policy_paths
scenarios ||--o{ shocks
scenarios ||--o{ runs
runs ||--o{ outputs
@enduml
```

---

3) Modeling Framework (QTC-centric, monthly step)

State variables (per month t):

- CP_t = growth of productive credit (y/y or m/m annualized as configured)
- CA_t = growth of non-GDP/asset credit
- Y_t = real GDP growth (q/q annualized, mapped to months via interpolation)
- pi_t = CPI inflation (y/y)
- H_t = house price inflation (y/y MHPI)
- E_t = equity return (KLCI y/y or m/m)
- NPL_t = NPL ratio
- External controls: export demand proxy X_t, terms of trade TOT_t, FX change FX_t.

Structural equations (parsimonious MVP)

1. Real activity Y_t = a0 + a1*CP_t* + *a2*CP_{t-1} + a3*X_t* + *a4*TOT_t + u_t

2. Inflation pi_t = b0 + b1*(CP_t - CP_star) + b2*FX_t* + *b3*cost_shock_t + v_t where CP_star is calibrated neutral productive-credit growth.

3. House prices H_t = c0 + c1*CA_t* + *c2*LTV_ease_t + c3*H_{t-1} + e_t

4. Equities E_t = d0 + d1*CA_t* + *d2*earnings_proxy_t + d3*global_eq_t + w_t

5. NPL dynamics Delta NPL_t = g0 + g1*(misalloc_t) + g2*min(0, H_{t-k}) + g3*Y_{t-k} + eta_t where misalloc_t = CA_t / (CA_t + CP_t) and k is a calibration lag.

Estimation: start with ARDL/Distributed-lag OLS for transparency; option to switch to state-space/TVP once history accumulates. Back-tests validate signs and lag structures.

---

4) Policy Mechanics (Quantitative Credit Guidance + Macroprudential)

4.1 Window-Guidance Allocator (per sector & purpose)

Inputs per month t: last month stock_amt, policy growth caps/paths by (sector,purpose), demand proxy (baseline credit path), and rule toggles.

Pseudo-algorithm:

```
for each (sector,purpose):
  allowed_growth = min(policy_cap_growth, baseline_demand_growth)
  target_flow = stock_amt * allowed_growth
apply macroprudential penalties:
  if purpose == NON_GDP_ASSET and (LTV_cap tightened or exposure cap hit):
      target_flow *= penalty_factor  # e.g., 0.6
reconcile system-wide constraints:
  scale flows to hit system total if needed (priority weights favor
PRODUCTIVE_GDP)
update credit_flow[sector,purpose]
```

Priority rule: When total supply is binding, allocate first to PRODUCTIVE_GDP, then remaining to NON_GDP_ASSET. Sector weights configurable.

4.2 Macroprudential levers feed into equations via shifters:

- LTV_ease_t from LTV/DTI settings → raises/lowers effective demand for mortgages.
- risk_weight_adj_bps shifts banks hurdle rates → lowers allowed_growth where tightened.
- policy_rate enters as secondary via small elasticities in CP_t and CA_t (default near zero).

4.3 Shocks

- Property price shock → exogenous H_t impulse + feedback to CA_t (wealth/credit channel).
- Export demand/commodity shock → X_t, TOT_t impulses.
- FX shock → pass-through to pi_t and potentially to NPL_t via FX debt share.

---

5) Calibration & Baselines

- Sample window (assumed): 2010–2019 baseline (pre-pandemic), with 2020–2022 down-weighted; rolling updates thereafter.
- Parameter estimation: OLS/ARDL with HAC errors; cross-validation on expanding windows; compare to simple elasticities.
- Neutral levels: CP_star estimated as HP-filtered trend of productive-credit growth.
- Validation: hit signs and elasticities expected by QTC (e.g., increase in CA_t → higher H_t with short lags; increase in CP_t → higher Y_t, modest effect on pi_t).

---

6) UI/Workflow (MVP)

- Scenario Builder: set sector/purpose credit paths, LTV/DTI caps, risk-weights, rate path, shocks.
- Run & Compare: execute baseline vs policy; show deltas for GDP, CPI, property, equity, NPLs; highlight credit composition share.
- Explainability: contributions (Shapley-style decomposition) by lever to each outcome; downloadable runbook (PDF).

---

7) Technology Choices (pinned for 2025)

- Backend: Python 3.11+, FastAPI \~0.116 (ASGI, OpenAPI docs); uvicorn, pydantic, orjson.
- Modeling: pandas, statsmodels 0.15+ (ARDL, statespace), numpy, scipy.
- DB: PostgreSQL 17 with TimescaleDB 2.21+ for hypertables, compression, retention.
- Auth: Keycloak 26.3+ (OIDC/SAML, passkeys/WebAuthn ready); RBAC via realm roles.
- Frontend: React 19, Vite/Next (CSR for on-prem), charts via Apache ECharts 6.
- Packaging/CI: Docker images; SBOM, Trivy scans; signed artifacts; deterministic builds.

---

8) Comparable Systems (for validation of approach)

- Bank of England System-Wide Exploratory Scenario (SWES) — system-level behavior under market shock with non-bank & bank interactions; informs dashboard comparison views.
- IMF FSAP stress-testing toolkits — solvency/liquidity stress frameworks; inform our NPL & liquidity-at-risk extensions.

---

9) Sequence of a Run

```
@startuml
actor Analyst
Analyst -> UI : Configure scenario & policies
UI -> API : POST /runs
API -> ME : Start(run_id)
ME -> TS : read series_data, credit_flow, policy_paths, shocks
```

```
ME -> ME : simulate monthly t..t+12 (allocator -> equations)
ME -> TS : write outputs (per metric/date)
ME -> OS : save artifacts (params, logs)
ME -> API : status=finished
API -> UI : results + confidence bands
@enduml
```

Implementation

A) Infrastructure & Ops (BNM on-prem/private cloud)

- **Runtime**: Kubernetes 1.29+ (3-node control plane, 4–6 worker nodes), Containerd, Calico; offline registry mirror.
- **Ingress**: NGINX Ingress + mTLS (internal), WAF at perimeter.
- **Secrets**: HashiCorp Vault (KV v2, PKI) or KMS-backed sealed-secrets.
- **Observability**: Prometheus + Grafana, Loki logs; OpenTelemetry auto-instrumentation for FastAPI.
- **Artifacts**: Private OCI registry; MinIO/S3 for run artifacts.
- **Backups**: pgBackRest for Postgres; object-store lifecycle rules.

Helm values (excerpt)

```
image:
  repository: registry.local/qtc-sim/api
  tag: v1.0.0
resources:
  requests: { cpu: "250m", memory: "512Mi" }
  limits:   { cpu: "2",    memory: "2Gi" }
ingress:
  tls: true
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/proxy-body-size: 20m
```

B) Database Setup (PostgreSQL 17 + TimescaleDB 2.21)

Extensions

```
CREATE EXTENSION IF NOT EXISTS timescaledb;
CREATE EXTENSION IF NOT EXISTS pgcrypto; -- for UUID gen
```

Core DDL (abridged)

```sql
CREATE TYPE purpose_enum AS ENUM ('PRODUCTIVE_GDP','NON_GDP_ASSET');
CREATE TYPE sector_enum  AS ENUM
('AGRI','MANUF','SERV','CONST_RE','SME','HH_MORT','HH_CONS','CORP_SEC_CRE');
CREATE TYPE freq_enum    AS ENUM ('D','W','M','Q','A');
CREATE TYPE metric_enum  AS ENUM
('GDP_REAL_GROWTH','CPI_INFL','HOUSE_PRICE_INFL','EQUITY_RET','NPL_RATIO','CRED_PROD_GROWTH','CRE

CREATE TABLE series_catalog (
  series_id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  source text NOT NULL,
  code   text,
  name   text,
  unit   text,
  freq   freq_enum NOT NULL,
  sector sector_enum,
  purpose purpose_enum,
  meta jsonb DEFAULT '{}'::jsonb,
  active boolean DEFAULT true
);

CREATE TABLE series_data (
  series_id uuid NOT NULL REFERENCES series_catalog(series_id),
  date date NOT NULL,
  value numeric NOT NULL,
  rev_id bigint,
  ingested_at timestamptz DEFAULT now(),
  checksum text,
  PRIMARY KEY(series_id,date)
);
SELECT create_hypertable('series_data','date', if_not_exists=>true);

CREATE TABLE credit_flow (
  date date NOT NULL,
  sector sector_enum NOT NULL,
  purpose purpose_enum NOT NULL,
  flow_amt numeric,
  stock_amt numeric,
  PRIMARY KEY(date,sector,purpose)
);
SELECT create_hypertable('credit_flow','date', if_not_exists=>true);

CREATE TABLE scenarios (
  scenario_id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  name text NOT NULL,
  created_by text NOT NULL,
  created_at timestamptz DEFAULT now(),
  notes text
```

```
);

CREATE TABLE policy_paths (
  scenario_id uuid NOT NULL REFERENCES scenarios(scenario_id) ON DELETE CASCADE,
  date date NOT NULL,
  sector sector_enum,
  purpose purpose_enum,
  credit_cap_growth_pct numeric,
  ltv_cap numeric,
  dti_cap numeric,
  risk_weight_adj_bps int,
  policy_rate numeric,
  fx_borrowing_cap numeric,
  flags jsonb DEFAULT '{}'::jsonb,
  PRIMARY KEY(scenario_id,date)
);

CREATE TABLE shocks (
  scenario_id uuid NOT NULL REFERENCES scenarios(scenario_id) ON DELETE CASCADE,
  date date NOT NULL,
  shock_type text NOT NULL,
  magnitude numeric NOT NULL,
  details jsonb DEFAULT '{}'::jsonb,
  PRIMARY KEY(scenario_id,date,shock_type)
);

CREATE TABLE runs (
  run_id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  scenario_id uuid NOT NULL REFERENCES scenarios(scenario_id) ON DELETE CASCADE,
  seed int,
  status text,
  started_at timestamptz,
  finished_at timestamptz,
  engine_version text,
  params jsonb,
  hash text
);

CREATE TABLE outputs (
  run_id uuid NOT NULL REFERENCES runs(run_id) ON DELETE CASCADE,
  date date NOT NULL,
  metric metric_enum NOT NULL,
  value numeric NOT NULL,
  band_lo numeric,
  band_hi numeric,
  meta jsonb DEFAULT '{}'::jsonb,
  PRIMARY KEY(run_id,date,metric)
);
```

Policies

- Row-level security on `scenarios`, `runs`, `outputs` by user/role.
- Timescale **compression** on `series_data`, `outputs`; retention for intermediates > 24 months.

C) Data Ingestion Pipelines

Connectors (batch)

1. **BNM OpenAPI** → monetary, banking, credit aggregates.
2. **OpenDOSM** → CPI, GDP; metadata & series IDs.
3. **NAPIC/JPPH (MHPI)** → quarterly MHPI; scrape/API/CSV depending on access; map to monthly via interpolation.
4. **Bursa/KLCI** → index levels via licensed vendor or manual CSV.

Mapping

- Normalize to monthly date index; tag `sector` & `purpose` using rules (e.g., mortgages → `HH_MORT` & `NON_GDP_ASSET`).
- Build `credit_flow` from stock/flow tables and classification logic.

ETL skeleton (Python)

```
import httpx, pandas as pd
from sqlalchemy import create_engine

BNM = 'https://apikijangportal.bnm.gov.my/openapi/...'
DOSM = 'https://developer.data.gov.my/static-api/opendosm/...'

def load_series(catalog_row):
    url = catalog_row['meta']['url']
    r = httpx.get(url, timeout=30)
    df = pd.DataFrame(r.json()['data'])
    # normalize columns -> date,value
    return df

# classification example
RULES = [
    {"contains":"mortgage", "sector":"HH_MORT", "purpose":"NON_GDP_ASSET"},
    {"contains":"working capital", "sector":"SME", "purpose":"PRODUCTIVE_GDP"},
]
```

D) Model Engine (FastAPI worker + Statsmodels)

Core steps

1. Build **baseline paths** for CP_t and CA_t from `credit_flow` and `policy_paths`.
2. Apply **window-guidance allocator** to compute target flows by `(sector,purpose)`.
3. Aggregate to CP_t / CA_t → feed structural equations.
4. Solve month-by-month t..t+12; write to `outputs` with confidence bands.

Engine interface

```
# app/engine/run_sim.py
from pydantic import BaseModel
class RunRequest(BaseModel):
    scenario_id: str
    horizon_months: int = 12
    seed: int = 42

class RunStatus(BaseModel):
    run_id: str
    status: str  # pending|running|finished|failed

# POST /runs -> enqueue
# GET /runs/{id} -> status & results link
```

Allocator (sketch)

```
def allocate_flows(stock, baseline, policy, macro):
    flows = {}
    for key in stock.keys():  # (sector,purpose)
        cap = min(policy[key].growth_cap, baseline[key].growth)
        penalty = 1.0
        if key[1] == 'NON_GDP_ASSET' and macro['ltv_tight']:
            penalty *= 0.6
        flows[key] = stock[key] * cap * penalty
    # prioritize PRODUCTIVE_GDP if system cap binds
    return reconcile_system_total(flows, priority='PRODUCTIVE_GDP')
```

Equations (statsmodels ARDL)

```
import statsmodels.api as sm
# example: H_t ~ CA_t + LTV_ease_t + H_{t-1}
model = sm.tsa.ARDL(endog=H, lags=1, exog=pd.DataFrame({"CA":CA,"LTV":LTV}))
```

```
res = model.fit()
forecast = res.predict(start=t0, end=t0+h)
```

E) API Surface (abridged)

```
POST   /scenarios                      -> create
GET    /scenarios/{id}                 -> detail
PUT    /scenarios/{id}                 -> update (if owner)
POST   /scenarios/{id}/policy_paths    -> upsert time-path
POST   /scenarios/{id}/shocks          -> upsert shocks
POST   /runs                           -> start run {scenario_id, horizon}
GET    /runs/{id}                      -> status
GET    /runs/{id}/outputs?metrics=...  -> time-series JSON/CSV
GET    /catalog/series                 -> list external series
POST   /catalog/series/{id}/sync       -> pull latest
```

Auth: OIDC via Keycloak (realm roles: `analyst`, `reviewer`, `admin`).

F) Frontend (React 19 + ECharts 6)

- **Pages**: Home (monitoring), Scenario Builder, Run Compare, Data Catalog, Admin.
- **Widgets**: Sector/purpose matrix editor, policy sliders, shocks panel, run queue, charts with band overlays.
- **UX**: two-pane layout; left = controls, right = charts; quick “delta vs baseline” badges.

G) Security & Compliance

- **RBAC & SSO** via Keycloak; passkeys enabled; session max-age policy.
- **Audit** on create/update/run/export; immutable log stream to Loki & WORM storage.
- **Data residency**: restrict egress; mirror public datasets to staging S3; cron sync via approved gateway.
- **PII**: none in v1; document DPIA nonetheless.

H) Testing & Validation

- **Unit**: allocator math, equation transforms, API contracts.
- **Back-test**: expanding-window forecast for 2015–2019; report RMSE/MAE.
- **Policy sanity**: scripted tests (tighten LTV → ↓ H_t within band; raise CP_t → ↑ Y_t, small $\Delta\pi_t$).
- **Performance**: SLA checks (single run $\leq$10 s; compare-two $\leq$20 s) with synthetic fixtures.

I) Runbook (Ops)

- **Deploy**: Helm install order → DB → Keycloak → API/Engine → UI.
- **Rotate**: DB credentials quarterly; rotate Keycloak signing keys annually.
- **Backup**: daily WAL archiving; weekly full; quarterly restore drills.
- **Incident**: model failure → auto rollback to last good `engine_version`; freeze new runs; alert oncall.

Milestones

Assumptions: Vanilla Kubernetes + Helm; Keycloak 26.3.x as **broker** to BNM IdP; on-prem private cloud; monthly data cadence. Timeline below is indicative (≈22 weeks total) and can compress/expand by team size.

Phase	Weeks	Scope & Deliverables	Exit Gate
0. Kickoff & Sec/ Infra Readiness	0–2	Project charter; risk register; **K8s namespaces**; private registry; Vault/KMS; network ACLs; **Keycloak realm** + IdP federation plan	**G0**: Sec/Infra checklist signed
1. Data Layer	2–6	PostgreSQL+Timescale provisioned; DDL applied; **series_catalog** seeded; BNM/DOSM connectors; MHPI & KLCI CSV loader; mapping to `credit_flow`; unit tests	**G1**: ≥90% required series ingested & validated
2. Model Engine MVP	5–10	Allocator (window-guidance); ARDL baselines (Y, π, H, E, NPL); scenario store; run orchestration; reproducibility hash	**G2**: Back-test signs correct; MAE within spec
3. Frontend MVP	8–12	Scenario Builder; Run Compare; charts w/ bands; CSV/PDF export; basic explainability view	**G3**: UX walkthrough approved
4. Calibration & Validation	10–14	2010–2019 baseline; 2020–2022 down-weight; sensitivity & shock suites; documentation	**G4**: Validation report accepted
5. Security & Performance	12–16	**Keycloak broker** to BNM IdP; RBAC roles; audit logs to Loki; perf tuning to hit ≤**10 s** per run; backup policies	**G5**: Pen-test & perf SLA pass
6. UAT & Pilot	16–20	Pilot users (Monetary Policy & Financial Stability); playbook; training; issues triage	**G6**: UAT sign-off
7. Go-Live (Controlled)	20–22	Prod deploy; runbook; DR drill; monitoring dashboards	**G7**: Go-live checklist green
8. v1.1 Enhancements (Post-GA)	+4–8	Bank-group granularity; monitoring mode & alerts; read-only API; model TVP option; ops hardening	Roadmap agrees

RACI (lightweight)

- **PO** (BNM sponsor): scope & sign-offs
- **Tech Lead** (vendor): architecture, delivery
- **Data Lead** (BNM): sources, mappings, validation
- **Security Lead** (BNM): identity, audit, reviews

Dependencies/Risks

- Access to MHPI detailed series; mitigation: accept quarterly headline + interpolation.
- IdP federation approval lead time; mitigation: start Phase 0 early with Security.
- Licensed KLCI data; mitigation: vendor or delayed import via CSV.

Gathering Results

A) Success Criteria & KPIs

- Model validity: Back-test MAE/RMSE vs 2015–2019 below thresholds (per metric); correct signs and lags per QTC (documented tests).
- Policy sensitivity: Tighten LTV by X → $\Delta H_t < 0$ within 3–6 months; increase CP_t path by Y → $\Delta Y_t > 0$ with muted $\Delta \pi_t$.
- Usability: UAT SUS score ≥ 75; scenario creation ≤ 10 minutes; export success rate 100%.
- Ops: $\geq 99.5\%$ availability in pilot; run time ≤ 10 s for 12-month horizon.
- Governance: 100% runs reproducible from stored scenario + engine hash; audit coverage on all create/update/run/export events.

B) Pilot Plan (confirmed)

- Pilot users: BNM Monetary Policy & Financial Stability teams.
- UAT window: 4 weeks (Weeks 16–20).
- Cadence: Weekly review (1 hr); issues triaged in 24 hrs; acceptance at G6.

C) Evaluation Workflow

1. Baseline validation: freeze calibration; run back-tests; publish report.
2. Policy suites: execute scripted scenarios (e.g., +5pp cut to non-GDP credit caps; −10pp LTV) and record deltas across KPIs.
3. Sensitivity matrix: local perturbations to each lever; produce tornado charts.
4. Stress day: combined export/FX/property shocks; verify stability and performance.
5. Sign-off: PO + Security + Data leads approve go-live.

D) Post-Production Monitoring

- Dashboards: latest credit composition, early-warning bands (non-GDP share, NPL momentum), run queue health.
- Alert rules (examples):
- Non-GDP credit share > 55% for 3 months → flag bubble risk rising.
- House-price inflation > 8% y/y and CA_t growing > 12% y/y → tighten LTV recommendation.

- NPL momentum > +0.3pp over 6 months → highlight sectoral misallocation.
- Model review: quarterly recalibration window; annual governance review; migration plan to TVP/state-space as data accumulate.

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)

Appendix — Sample CSV Templates & Seed Scenario

> **How to use (UI path)**: 1) Go to **Data Catalog → Upload CSV** for `credit_flow_seed.csv` (optional if you already have data). 2) Open **Scenario Builder → Import policy paths** using `policy_paths_seed.csv`. 3) (Optional) Add `shocks_seed.csv`. 4) Click **Run** and compare against baseline.
>
> **How to use (API)**: endpoints are in *Implementation → E) API Surface*. Example `curl` at the end of this appendix.

A) CSV Templates

1) `policy_paths_seed.csv`

- **Purpose**: sets **quantitative credit guidance** (growth caps) and macroprudential levers by `(sector,purpose)` and month.
- **Units/format**: `date` = `YYYY-MM-01`; `credit_cap_growth_pct` = **monthly rate as decimal** (e.g., `0.008` ≈ 0.8% m/m ≈ 10% annualized). `ltv_cap` as percent (e.g., 0.70 = 70%). Empty cells = unchanged.

```
scenario_name,date,sector,purpose,credit_cap_growth_pct,ltv_cap,dti_cap,risk_weight_adj_bps,polic
Baseline-2025Q1,2025-01-01,SME,PRODUCTIVE_GDP,0.010,,,0,,
Baseline-2025Q1,2025-01-01,HH_MORT,NON_GDP_ASSET,0.004,0.70,0.40,50,,
Baseline-2025Q1,2025-01-01,MANUF,PRODUCTIVE_GDP,0.009,,,0,,
Baseline-2025Q1,2025-02-01,SME,PRODUCTIVE_GDP,0.010,,,0,,
Baseline-2025Q1,2025-02-01,HH_MORT,NON_GDP_ASSET,0.004,0.70,0.40,50,,
Baseline-2025Q1,2025-02-01,MANUF,PRODUCTIVE_GDP,0.009,,,0,,
Baseline-2025Q1,2025-03-01,SME,PRODUCTIVE_GDP,0.010,,,0,,
Baseline-2025Q1,2025-03-01,HH_MORT,NON_GDP_ASSET,0.004,0.70,0.40,50,,
Baseline-2025Q1,2025-03-01,MANUF,PRODUCTIVE_GDP,0.009,,,0,,
```

> Tip: For a full 12-month horizon, copy/paste the three monthly rows forward and adjust as needed. Add additional sectors as required (`AGRI`, `SERV`, `CONST_RE`, `SME`, `HH_CONS`, `CORP_SEC_CRE`). `purpose` must be `PRODUCTIVE_GDP` or `NON_GDP_ASSET`.

2) `shocks_seed.csv` **(optional)**

- **Purpose**: inject exogenous shocks for experiments.
- **Conventions**: `magnitude` is a decimal change (e.g., `-0.03` = −3%). `details_json` can be `{}`.

```
scenario_name,date,shock_type,magnitude,details_json
Baseline-2025Q1,2025-06-01,PROPERTY_PRICE,-0.03,"{}"
Baseline-2025Q1,2025-04-01,EXPORT_DEMAND,0.02,"{\"notes\":\"mild rebound\"}"
```

3) `credit_flow_seed.csv` **(optional for demo without external data)**

- **Purpose**: provides **starting stocks** and a few months of **flows** by sector/purpose if you don't yet ingest from BNM.
- **Units/format**: MYR **millions**. Provide one **stock row** for the anchor month (e.g., 2024-12-01), then monthly **flow rows** thereafter. The system derives subsequent stocks.

```
date,sector,purpose,stock_amt,flow_amt
2024-12-01,SME,PRODUCTIVE_GDP,100000,
2024-12-01,HH_MORT,NON_GDP_ASSET,450000,
2025-01-01,SME,PRODUCTIVE_GDP,,1000
2025-01-01,HH_MORT,NON_GDP_ASSET,,1800
2025-02-01,SME,PRODUCTIVE_GDP,,1005
2025-02-01,HH_MORT,NON_GDP_ASSET,,1750
2025-03-01,SME,PRODUCTIVE_GDP,,1010
2025-03-01,HH_MORT,NON_GDP_ASSET,,1700
```

Tip: You can add more `(sector,purpose)` pairs and continue flows for 12 months.

B) Seed Scenario (JSON examples)

1) Create scenario

```
{
  "name": "Baseline-2025Q1",
  "created_by": "demo_user",
  "notes": "Baseline with productive-credit uplift and asset-credit restraint"
}
```

2) Example policy paths (JSON alternative to CSV)

```
[

{"date":"2025-01-01","sector":"SME","purpose":"PRODUCTIVE_GDP","credit_cap_growth_pct":
```

```
0.010},

{"date":"2025-01-01","sector":"HH_MORT","purpose":"NON_GDP_ASSET","credit_cap_growth_pct":
0.004,"ltv_cap":0.70,"dti_cap":0.40,"risk_weight_adj_bps":50},

{"date":"2025-01-01","sector":"MANUF","purpose":"PRODUCTIVE_GDP","credit_cap_growth_pct":
0.009}
]
```

3) Example shocks

```
[

{"date":"2025-06-01","shock_type":"PROPERTY_PRICE","magnitude":-0.03,"details":
{}},
  {"date":"2025-04-01","shock_type":"EXPORT_DEMAND","magnitude":0.02,"details":
{"notes":"mild rebound"}}
]
```

C) Quickstart (API `curl`)

Replace `BASE` with your API URL and set a valid `Bearer` token.

```
# 1) Create the scenario
SCENARIO_ID=$(curl -s -X POST "$BASE/scenarios"
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json"
  -d '{"name":"Baseline-2025Q1","created_by":"demo_user","notes":"Baseline with
productive-credit uplift and asset-credit restraint"}'
  | jq -r .scenario_id)

echo "Scenario: $SCENARIO_ID"

# 2) Upload policy paths from CSV
curl -s -X POST "$BASE/scenarios/$SCENARIO_ID/policy_paths"
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: text/csv"
  --data-binary @policy_paths_seed.csv

# 3) (Optional) Upload shocks from CSV
curl -s -X POST "$BASE/scenarios/$SCENARIO_ID/shocks"
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: text/csv"
  --data-binary @shocks_seed.csv

# 4) (Optional) Seed credit flows
```

```
curl -s -X POST "$BASE/catalog/series/credit_flow_seed:import"
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: text/csv"
  --data-binary @credit_flow_seed.csv

# 5) Start a 12-month run
RUN_ID=$(curl -s -X POST "$BASE/runs"
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json"
  -d "{\"scenario_id\":\"$SCENARIO_ID\",\"horizon_months\":12}" | jq -r .run_id)

echo "Run: $RUN_ID"

# 6) Fetch results (GDP and Property index)
curl -s "$BASE/runs/$RUN_ID/outputs?metrics=GDP_REAL_GROWTH,HOUSE_PRICE_INFL"
  -H "Authorization: Bearer $TOKEN" | jq .
```

Note: Endpoint `/catalog/series/credit_flow_seed:import` is a convenience import one-liner—implement as a small controller that maps the CSV to `credit_flow` rows for demo purposes.

Appendix — Python Loader Script (Local Laptop)

Minimal client to: create scenario → upload CSVs → start run → poll → save outputs.

`qtc_loader.py`

```
#!/usr/bin/env python3
import argparse, os, sys, time, json
from typing import Optional
import requests

DEF_BASE = os.getenv("QTC_BASE_URL", "https://qtc.local/api")
DEF_TOKEN = os.getenv("QTC_TOKEN", "")
DEF_CA = os.getenv("QTC_CA_BUNDLE")  # e.g., /path/to/bnm-root-ca.pem

SESSION = requests.Session()

class Api:
    def __init__(self, base: str, token: str, verify: Optional[str|bool]=True):
        self.base = base.rstrip('/')
        self.h = {"Authorization": f"Bearer {token}"} if token else {}
        self.verify = verify

    def _url(self, path: str) -> str:
        return f"{self.base}{path}"

    def post_json(self, path: str, payload: dict):
        r = SESSION.post(self._url(path), headers={**self.h, "Content-Type":
```

```
"application/json"},
                           data=json.dumps(payload), timeout=60,
verify=self.verify)
        _ok(r)
        return r.json()

    def post_csv(self, path: str, csv_path: str):
        with open(csv_path, 'rb') as f:
            r = SESSION.post(self._url(path), headers={**self.h, "Content-Type":
"text/csv"},
                             data=f, timeout=120, verify=self.verify)
        _ok(r)
        return r.json() if r.text and r.headers.get('content-
type','').startswith('application/json') else {"status": r.status_code}

    def get_json(self, path: str, params: dict | None = None):
        r = SESSION.get(self._url(path), headers=self.h, params=params,
timeout=60, verify=self.verify)
        _ok(r)
        return r.json()


def _ok(r: requests.Response):
    if r.status_code >= 400:
        try:
            msg = r.json()
        except Exception:
            msg = r.text
        raise SystemExit(f"HTTP {r.status_code}: {msg}")


def create_scenario(api: Api, name: str, created_by: str, notes: str) -> str:
    resp = api.post_json('/scenarios', {"name": name, "created_by": created_by,
"notes": notes})
    sid = resp.get('scenario_id') or resp.get('id')
    if not sid:
        raise SystemExit("No scenario_id in response")
    print(f"Scenario created: {sid}")
    return sid


def upload_policy(api: Api, scenario_id: str, csv_path: str):
    print(f"Uploading policy paths: {csv_path}")
    api.post_csv(f"/scenarios/{scenario_id}/policy_paths", csv_path)


def upload_shocks(api: Api, scenario_id: str, csv_path: str):
    print(f"Uploading shocks: {csv_path}")
```

```
    api.post_csv(f"/scenarios/{scenario_id}/shocks", csv_path)


def seed_credit(api: Api, csv_path: str):
    print(f"Seeding credit flows: {csv_path}")
    # Preferred convenience endpoint from spec; fallback if not implemented
    try:
        api.post_csv("/catalog/series/credit_flow_seed:import", csv_path)
    except SystemExit as e:
        if "404" in str(e):
            api.post_csv("/credit_flow/import", csv_path)
        else:
            raise


def start_run(api: Api, scenario_id: str, horizon: int) -> str:
    resp = api.post_json('/runs', {"scenario_id": scenario_id, "horizon_months":
horizon})
    rid = resp.get('run_id') or resp.get('id')
    if not rid:
        raise SystemExit("No run_id in response")
    print(f"Run started: {rid}")
    return rid


def poll_run(api: Api, run_id: str, timeout_s: int = 300, interval_s: int = 3) -
> str:
    deadline = time.time() + timeout_s
    while time.time() < deadline:
        st = api.get_json(f"/runs/{run_id}")
        status = (st.get('status') or st.get('state') or '').lower()
        if status in {"finished", "failed"}:
            print(f"Run status: {status}")
            return status
        time.sleep(interval_s)
    raise SystemExit("Timeout waiting for run to finish")


def fetch_outputs(api: Api, run_id: str, metrics: list[str], out_csv: str):
    params = {"metrics": ",".join(metrics)} if metrics else {}
    data = api.get_json(f"/runs/{run_id}/outputs", params=params)
    # Expecting list of {date, metric, value, ...}
    import csv
    rows = data if isinstance(data, list) else data.get('rows', [])
    if not rows:
        print("Warning: no rows returned")
    fields = sorted({k for r in rows for k in r.keys()}) or
["date","metric","value"]
```

```
    with open(out_csv, 'w', newline='') as f:
        w = csv.DictWriter(f, fieldnames=fields)
        w.writeheader()
        for r in rows:
            w.writerow(r)
    print(f"Saved outputs → {out_csv}")


def main():
    p = argparse.ArgumentParser(description="QTC Simulator loader")
    p.add_argument('--base', default=DEF_BASE, help='Base API URL (or
QTC_BASE_URL env)')
    p.add_argument('--token', default=DEF_TOKEN, help='Bearer token (or
QTC_TOKEN env)')
    p.add_argument('--ca', default=DEF_CA, help='Custom CA bundle path for TLS
(or QTC_CA_BUNDLE env)')
    p.add_argument('--scenario', default='Baseline-2025H1')
    p.add_argument('--created-by', default=os.getenv('USER','demo_user'))
    p.add_argument('--notes', default='Baseline with productive-credit uplift
and asset-credit restraint')
    p.add_argument('--policy', default='policy_paths_seed.csv')
    p.add_argument('--shocks', default=None, help='Optional shocks CSV')
    p.add_argument('--credit', default=None, help='Optional credit_flow CSV')
    p.add_argument('--horizon', type=int, default=12)
    p.add_argument('--metrics',
default='GDP_REAL_GROWTH,HOUSE_PRICE_INFL,NPL_RATIO')
    p.add_argument('--out', default='outputs.csv')
    args = p.parse_args()

    verify = True if not args.ca else args.ca
    api = Api(args.base, args.token, verify=verify)

    sid = create_scenario(api, args.scenario, args.created_by, args.notes)
    if args.policy and os.path.exists(args.policy):
        upload_policy(api, sid, args.policy)
    if args.shocks and os.path.exists(args.shocks):
        upload_shocks(api, sid, args.shocks)
    if args.credit and os.path.exists(args.credit):
        seed_credit(api, args.credit)

    rid = start_run(api, sid, args.horizon)
    status = poll_run(api, rid)
    if status != 'finished':
        raise SystemExit("Run failed; check server logs")

    metrics = [m.strip() for m in args.metrics.split(',') if m.strip()]
    fetch_outputs(api, rid, metrics, args.out)
```

```
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(130)
```

`requirements.txt`

```
requests>=2.32
```

Usage

```
python3 -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt
export QTC_BASE_URL="https://qtc.local/api"   # or http://localhost:8000
export QTC_TOKEN="<paste your bearer token>"
# Optional: export QTC_CA_BUNDLE="/path/to/bnm-root-ca.pem"
python qtc_loader.py
  --scenario Baseline-2025H1
  --policy policy_paths_seed.csv
  --shocks shocks_seed.csv
  --credit credit_flow_seed.csv
  --out outputs.csv
```

Notes

- If your on-prem TLS uses a private CA, pass `--ca /path/to/ca.pem` or set `QTC_CA_BUNDLE`.
- The client retries only at the HTTP layer via exceptions. If you expect 429s, consider adding a backoff wrapper.
- Endpoint names align with this spec; adjust if your implementation differs.

Appendix — FastAPI Stub (Local Dev)

> Minimal server implementing the endpoints in this spec so you can run the full flow locally. Stores everything **in memory** and returns **synthetic outputs** based on your policy/shock inputs. Swap out the stubbed model with the real engine when ready.

`main.py`

```
from __future__ import annotations
import csv
import io
import uuid
```

```python
from datetime import date, datetime
from typing import Dict, List, Optional

from fastapi import FastAPI, HTTPException, Request, BackgroundTasks
from fastapi.responses import JSONResponse
from pydantic import BaseModel

app = FastAPI(title="QTC Simulator Stub", version="1.0.0")

# ------------------------------
# In-memory stores (demo only)
# ------------------------------
SCENARIOS: Dict[str, dict] = {}
POLICY_PATHS: Dict[str, List[dict]] = {}
SHOCKS: Dict[str, List[dict]] = {}
CREDIT_FLOW: List[dict] = []  # optional demo data
RUNS: Dict[str, dict] = {}
OUTPUTS: Dict[s
```