



RECORA

Domain-agnostic recommendation engine

ISSUED BY

Multitv Solution Pvt Ltd

Technical Team

Siddarth Samant

Vikas Sharma

Summary

A step-by-step playbook to build and run a domain-agnostic recommendation engine (works for OTT, Pharma, Consulting, Events, Retail...) using Golang + MongoDB + Redis. It's broken into phases; each step has *what to build*, *where it lives*, and *the acceptance checks* so a junior dev can execute.

Phase 0 — Core contracts (1 day)

1. Canonical Schemas (stable across domains)

Item

```
{
  "tenant_id": "<string>",
  "item_id": "<string>",
  "type": "content|product|service|event|expert|...",
  "text": {"title": "...", "desc": "..."},
  "facets": {"genre": "...", "industry": "...", "lang": "...", "city": "...", "series_id": "..."},
  "duration_seconds": 0,
  "availability": {"geo": ["IN"], "device": ["android", "ios", "web"], "time": {"start": "...", "end": "..."}},
  "compliance": {"age_18_plus": false, "hcp_only": false, "on_label": true}
}
```

Interaction

```
{
  "tenant_id": "<string>",
  "user_id": "<string>",
  "item_id": "<string|null>",
  "event_type": "view|play|detail|download|add_to_cart|purchase|register|attend|contact_submit|search",
  "value": 1,
  "ts": "2025-08-19T10:00:00Z",
  "session_id": "<optional>",
  "context": {"device": "android", "geo": "IN", "role": "...", "city": "...", "referrer": "..."}
}
```

1) tenants

Purpose: onboard a client + hold HMAC keys (rotatable).

Doc

```
{
  "tenant_id": "acme",
  "name": "Acme Corp",
  "status": "active",
  "keys": [
    { "key_id": "k1", "hmac_secret": { "$binary": "...=" }, "active": true, "created_at": ISODate() },
    { "key_id": "k0", "hmac_secret": { "$binary": "...=" }, "active": false, "created_at": ISODate() }
  ],
  "created_at": ISODate(), "updated_at": ISODate()
}
```

Indexes

- { tenant_id: 1 } (unique)
-

2) apps

Purpose: per-tenant app/config (adapter, policies, scoring).

Doc

```
{
  "tenant_id": "acme",
  "app_id": "web",
  "name": "Web App",
  "status": "active",
  "adapter": "ott",           // maps raw→canonical
}
```

```
"policy_chain": ["age_gate","geo_device","dedup_series"],

"facet_weights": { "genre":1.0, "series_id":0.7, "lang":0.3 },

"scoring_weights": { "covisit":0.45, "trending":0.30, "content":0.15, "fresh_bonus":0.10,
"seen_penalty":0.10 },

"mmr_lambda": 0.7,

"retention_days": { "interactions":365, "audit":90 },

"limits": { "rpm": 600 },

"created_at": ISODate(), "updated_at": ISODate()

}
```

Indexes

- { tenant_id:1, app_id:1 } (unique)

3) idempotency_keys

Purpose: webhook dedupe.

Doc

```
{ "tenant_id":"acme", "app_id":"web", "idem_key":"evt-abc-123", "created_at": ISODate() }
```

Indexes

- { tenant_id:1, app_id:1, idem_key:1 } (unique)
- TTL on created_at → 48h

4) watermarks

Purpose: pull-ingestion checkpoints per stream.

Doc

```
{ "tenant_id": "acme", "app_id": "web", "name": "analytics_events", "last_ts": ISODate(),  
  "updated_at": ISODate() }
```

Indexes

- { tenant_id:1, app_id:1, name:1 } (unique)
-

5) reco_exposures

Purpose: log what we served for A/B and audit.

Doc

```
{  
  "tenant_id": "acme", "app_id": "web",  
  "exp_id": "reco_v1", "variant": "B",  
  "user_id": "U123", "request_id": "r-8f3c",  
  "context": "home", "limit": 30,  
  "served": [  
    { "c_id": "I11", "rank": 1, "score": 1.24, "why": { "covisit": 0.7, "trending": 0.3 } },  
    { "c_id": "I25", "rank": 2, "score": 1.18, "why": { "covisit": 0.6, "content": 0.4 } }  
  ],  
}
```

```
"req_ts": ISODate()  
}
```

Indexes

- { tenant_id:1, app_id:1, exp_id:1, req_ts:-1 }
- { tenant_id:1, app_id:1, user_id:1, req_ts:-1 }

6) `reco_outcomes` (optional mirror; otherwise join `interactions`)

Purpose: outcome events keyed by `request_id` for deterministic joins.

Doc

```
{  
  "tenant_id":"acme","app_id":"web",  
  "request_id":"r-8f3c","user_id":"U123","c_id":"I11",  
  "event_type":"click",      // click|play_60s|download|register|...  
  "ts": ISODate()  
}
```

Indexes

- { tenant_id:1, app_id:1, request_id:1, ts:-1 }

7) `daily_kpis`

Purpose: rolled-up experiment metrics.

Doc

```
{
  "tenant_id":"acme","app_id":"web","date":"2025-08-19",
  "exp_id":"reco_v1","variant":"B",
  "served":1240000,
  "ctr":0.128,"play60_rate":0.083,"goal_rate":0.012,
  "Avg_watch_sec":214.3,"diversity_at_30":0.78,"freshness_at_30":0.64,
  "generated_at": ISODate()
}
```

Indexes

- { tenant_id:1, app_id:1, date:1, exp_id:1, variant:1 } (unique)

8) reco_audit

Purpose: immutable “what & why” log for governance.

Doc

```
{
  "tenant_id":"acme","app_id":"web","user_id":"U123","request_id":"r-8f3c",
  "policies":["age_gate","geo_device","dedup_series"],
  "items":[{"c_id":"I11","score":1.24,"why":{"covisit":0.7,"trending":0.3}}],
  "ts": ISODate()
}
```

Indexes

- { tenant_id:1, app_id:1, ts:-1 }
- TTL on ts → e.g., 90d (per app config)

9) user_aliases

Purpose: map external identities; simplifies GDPR delete/export.

Doc

```
{  
  
  "tenant_id":"acme","app_id":"web",  
  
  "external_id":"email:hash@domain.com", // or phone:..., sso:sub  
  
  "u_id":"U123",  
  
  "created_at": ISODate()  
}
```

Indexes

```
{ tenant_id:1, app_id:1, external_id:1 } (unique)  
{ tenant_id:1, app_id:1, u_id:1 }
```

10) continue_watching

Purpose: user progress for “Continue Watching”.

Doc

```
{
```



```
"tenant_id":"acme","app_id":"web","u_id":"U123","c_id":"I456",
"progress_sec": 137, "duration_sec": 1800,
"updated_at": ISODate()
}
```

Indexes

```
{ tenant_id:1, app_id:1, u_id:1, c_id:1 } (unique)
{ tenant_id:1, app_id:1, u_id:1, updated_at:-1 }
```

11) `user_topn` (optional, from ALS/CF)

Purpose: per-user candidate cache for stronger personalization.

Doc

```
{
"tenant_id":"acme","app_id":"web","u_id":"U123",
"items":[{"c_id":"I9","score":2.3}, {"c_id":"I7","score":1.8}, ... ],
"updated_at": ISODate()
}
```

Indexes

- { tenant_id:1, app_id:1, u_id:1 } (unique)
- { tenant_id:1, app_id:1, updated_at:-1 }

12) `pair_counts` (optional, incremental covisit store)

Purpose: maintain rolling A→B weights before writing `covisit_topk`.

Doc

```
{
  "tenant_id":"acme","app_id":"web","src":"l11",
  "dst":"l25","w": 12.73,          // accumulated weight
  "updated_at": ISODate()
}
```

Indexes

- { tenant_id:1, app_id:1, src:1, dst:1 } (unique)
- { tenant_id:1, app_id:1, updated_at:-1 }

13) `content_embeddings` (optional, for two-tower/vector)

Purpose: store item/user embeddings for ANN/vector search.

Doc

```
{
  "tenant_id":"acme","app_id":"web",
  "kind":"item",          // item | user
  "id":"l11",             // c_id or u_id
  "vector":[0.012, -0.33, ...], // float32[]
  "updated_at": ISODate(),
  "meta":{"dim":128, "model":"tw-2025-08" }
}
```


Indexes

- `{ tenant_id:1, app_id:1, kind:1, id:1 }` (unique)
- `{ tenant_id:1, app_id:1, kind:1, updated_at:-1 }`

14) (Recap) Precompute outputs you already have

2. `popular_day: { tenant_id, app_id, day, items:[{c_id, score}], updated_at }`
Index: `{ tenant_id:1, app_id:1, day:1 }`
3. `covisit_topk: { tenant_id, app_id, src, sims:[{c_id, score}], updated_at }`
Index: `{ tenant_id:1, app_id:1, src:1 }`
4. `content_sim_topk`: same shape/index as above

5. Collections & Indexes (add `tenant_id` everywhere)

- `contents(tenant_id, item_id)[unique]`; facets indexes you'll filter on
 - `plays(tenant_id, u_id, ts-1) ; (tenant_id, c_id, ts-1) ; (tenant_id, u_id, c_id, ts-1)`
 - `popular_day(tenant_id, day)`
 - `covisit_topk(tenant_id, src)`
 - `content_sim_topk(tenant_id, src)`
 - `reco_logs(tenant_id, u_id, ts-1)`
 -  **Acceptance:** index creation runs at boot without error; CRUD smoke tests pass.
-

Phase 1 — Ingestion (Push or Pull) (1–2 days)

Collects all raw signals: items (content/products), user interactions (views, clicks, plays, downloads), and continues watching/progress. Normalizes them into a domain-agnostic schema `{tenant_id, app_id, user_id, item_id, event_type, ts}` stored in MongoDB.

3. Adapters (per domain, 100–150 LoC)

Interface:

```
type DomainAdapter interface {  
    MapItem(raw map[string]any) (Item, error)  
    MapInteraction(raw map[string]any) (Interaction, error)  
}
```

-
- Implement `OTTAdapter`, `PharmaAdapter`, `ConsultingAdapter` as needed.
- ☒ *Acceptance*: sample raw payload → mapped Item/Interaction saved in Mongo.

4. Push API (recommended)

- `POST /reco/ingest/item?tenant_id=T&adapter=A`
- `POST /reco/ingest/event?tenant_id=T&adapter=A`
- Add HMAC `X-Signature` and Idempotency-Key (dedup table with TTL 48h).
- ☒ *Acceptance*: replay same event → exactly one row in `plays`.

OR

5. Pull job (quick start)

- `cmd/ingest/pull_analytics.go`: poll your Analytics API with `?since=<watermark>&limit=....`
- Keep the watermark in `meta.watermarks`.

- ☒ *Acceptance*: restart the job → resumes from last watermark without dupes.
-

Phase 2 — Precompute pipelines (fast, offline) (2–3 days)

Runs batch jobs (cron) that crunch analytics into ready-to-use features:

Trending (most popular by recency)

Co-visit (items consumed together)

Content similarity (metadata overlap).

Outputs stored in `popular_day`, `covisit_topk`, `content_sim_topk` collections for fast lookup.

6. Trending (7-day, recency decay)

- Score per item: $\exp(-\alpha * \text{age_hours}) * \min(1, \text{play_seconds}/\text{duration})$; multiply by $\log(2 + \text{unique_users})$.
- Write `popular_day{tenant_id, day, items:[{c_id, score}]}`.
- Run every 30–60 minutes.
- ☒ *Acceptance*: top list looks sane; no timeouts; doc count bounded (1/day/tenant).

7. Co-visitation (item→item)

- Sessionize per user (gap \leq 45 min).
- For each session: pair items; weight = $1/(1+\Delta t_minutes)$ both directions.
- Aggregate across users → keep Top-K per `src` into `covisit_topk`.
- Run every 6–12 hours (14–30d lookback).
- ☒ *Acceptance*: neighbors are intuitive; each doc \leq K entries; job under SLA.

8. Content-similarity (metadata)

- Build sparse vectors from **facets** (configurable weights per tenant).
- Similarity = Jaccard or cosine; store Top-K into **content_sim_topk**.
- Run daily or on catalog changes.
- ☒ **Acceptance**: similar items share key facets; latency under SLA.

(Optional AI v1 later)

- ALS (Matrix Factorization) nightly: produce **user_topn{tenant_id,u_id,items[]}** to strengthen personalization.

Phase 3 — Online serving (low latency) (1–2 days)

9. Feature assembly (per candidate)

- **covisit**, **trending**, **content_sim**, **freshness_bonus**, **seen_recently_penalty**, plus simple context matches (e.g., lang/geo).
- Keep feature computation $O(1)$ via precompute reads + small lookups.

10. Scoring (start heuristic, upgrade later)

score = 0.45*covisit + 0.30*trending + 0.15*contentSim + 0.10*freshBonus - 0.10*seenPenalty

- Later: replaced with LTR (LightGBM) using the same features.

11. Diversity (MMR)

- Re-rank with $MMR(\lambda=0.7)$ using item-sim (from content sim or same series/industry).
- Avoid “row-locking” on one series/topic.

12. Policies (per tenant)

- Chain of filters: AgeGate, GeoDevice, TimeWindow, DedupSeries, HCPOnly/OnLabel (pharma), NDA/Conflict (consulting).
- ☒ *Acceptance*: unit tests show policy keeps/blocks expected items.

13. Caching

- Redis key: $reco:\{tenant\}:\{user\}:\{context\} \rightarrow$ list of item_ids (+ scores).
- TTL 5–15 min; invalidate on new interaction or at TTL.
- Fallback: if Redis/Mongo slow \rightarrow return Trending only.
- ☒ *Acceptance*: P95 < 60 ms @ limit=30; cache hit rate > 70%.

14. HTTP API

- $GET /reco/recommend?tenant_id=T\&user_id=U\&context=home\&limit=30$

Response:

```
{"user_id":"U","items":[{"c_id":"I1","score":1.23,"why":{"covisit":0.6,"trending":0.4}}]}
```

- $GET /reco/debug/why?...$ returns feature breakdown for a specific item (no secrets).

Phase 4 – A/B tests & analytics (1 day)

Buckets users into control/treatment groups, serving different algorithms/weights per variant.

Logs exposures and outcomes (clicks, plays, conversions) and computes daily KPIs (CTR, watch-time, diversity, freshness) to measure lift.

15. Experiment assignment

- Sticky assignment in Redis: `ab:reco:{tenant}:{user} → A|B|C`.
- Variants (e.g.):
A = Trending+Content; B = Covisit+Content+MMR; C = +ALS (if enabled).

16. KPI logging

- Log every recommendation & click/play outcome with variant.
- Daily KPIs per tenant:
 - CTR→Play/Download/Register/Contact
 - 60s completion / dwell
 - Hours watched / user (OTT) or Leads / 1k visits (consulting)
 - Diversity@N, Freshness@N, Rule-hit counts.
- ☒ **Acceptance:** daily report generated; anomalies flagged.

Phase 5 – Multi-tenant onboarding flow (2–3 days per domain)

Adds real ML: Matrix Factorization (ALS), Learning-to-Rank, or Two-Tower neural models. These models replace or blend with heuristics, giving deeper personalization while staying tenant/app-aware.

17. New tenant checklist

- Create `config/tenants/<tenant>.yaml`:

- adapter name, facet weights, policy list, scoring weights.
- Implement adapter (if new domain).
- Add required facet indexes (industry, topic, city...).

18. Data backfill

- Backfill **contents** (catalog); then backfill last 30–90d **interactions**.
- Run precomputers (trending, content-sim, co-visit).
- ☒ **Acceptance:** `/recommend` returns non-empty, relevant lists.

19. Go-live

- Wire client apps (web/mobile/CRM) to call `/recommend`.
- Monitor P95 latency, error rate, KPIs; keep playbooks for rollback.

Phase 6 — Reliability, security, governance (ongoing)

Wraps the engine with health checks, logging, rate limits, circuit breakers, and Redis job locks for safety.

Adds HMAC auth, key rotation, consent gates, audit logs, retention/TTL, metrics, alerts, and backups, making the system production-grade and multi-tenant compliant.

20. SLOs

- API P95 < 60 ms, error < 0.5%, availability 99.9%.
- Precomputes finish within the window (trending ≤ 5 min; covisit ≤ 45 min).

21. Security

- HMAC on ingestion endpoints; clock skew ± 5 min.
- Idempotency keys; rate limits on tenants.
- Hash user IDs if needed per tenant policy.

22. Governance & privacy

- Consent scopes on features; no PHI/PII stored beyond allowed IDs.
- Audit logs (pharma): store policy decisions + feature snapshot per serve.

23. Observability

- Structured logs with correlation IDs.
- Prometheus metrics (or your current stack): cache hit, Mongo/Redis latency, job duration, queue lag.
- Alerts on SLO breaches.

Phase 7 – “AI upgrades” (optional, after v1 proves value)

24. ALS (Matrix Factorization)

- Nightly train (Python `implicit` is fine), confidence = function(play time, completion, recency).
- Materialize `user_topn` & `item_factors` → boost candidates.

25. Learning-to-Rank (LightGBM)

- Train weekly on exposure → label data.
- Replace heuristic with model scores; keep MMR & policies.

26. Two-Tower + ANN (advanced)

- Train dual encoders; serve via Milvus/Qdrant (or Faiss service).
 - Use for first-stage retrieval; re-rank with LTR.
-

Cron/Systemd schedule (no Docker required)

- Trending: `*/30 * * * *`
 - Covisit: `0 */6 * * *`
 - Content-sim: `0 */12 * * *`
 - (ALS nightly if enabled)
-

Acceptance test suite (must pass before launch)

- Indexes created; collections seeded; health endpoint OK.
- Ingest Push/Pull produces deduped interactions; watermark resumes.
- Precomputes produce non-empty, bounded docs per tenant.
- `/recommend` P95 < 60 ms with cache warm and cold (fallback OK).
- Policies enforce correctly (pharma HCP/on-label; consulting NDA; OTT age/geo).
- A/B pipeline logs and daily KPIs computed.
- Security: HMAC verified, rate limit enforced, idempotency tested.
- Runbook written (backfill, rotate weights, clear caches, rollback).

Phase 1 — Ingestion (Push or Pull) (1-2 days)

1) MongoDB collections (and indexes)

Create once at startup:

```
// util/mongo_indexes.go
package util

import (
    "context"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

func EnsureIngestionIndexes(db *mongo.Database) error {
    // tenants
    _, _ = db.Collection("tenants").Indexes().CreateOne(context.TODO(),
        mongo.IndexModel{Keys: bson.D{{Key: "tenant_id", Value: 1}}, Options:
options.Index().SetUnique(true)})

    // items (catalog)
    _, _ = db.Collection("items").Indexes().CreateMany(context.TODO(), []mongo.IndexModel{
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "item_id", Value: 1}}, Options:
options.Index().SetUnique(true)},
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "type", Value: 1}}},
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "updated_at", Value: -1}}},
    })

    // interactions (events)
    _, _ = db.Collection("interactions").Indexes().CreateMany(context.TODO(),
[]mongo.IndexModel{
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "ts", Value: -1}}},
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "user_id", Value: 1}, {Key: "ts", Value:
-1}}},
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "item_id", Value: 1}, {Key: "ts", Value:
-1}}},
        {Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "event_type", Value: 1}, {Key: "ts",
Value: -1}}},
    })
}
```

```

    })

    // idempotency (webhook dedupe)
    _, _ = db.Collection("idempotency_keys").Indexes().CreateOne(context.TODO(),
        mongo.IndexModel{Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "idem_key",
Value: 1}}, Options: options.Index().SetUnique(true)}}

    // watermarks (pull mode)
    _, _ = db.Collection("watermarks").Indexes().CreateOne(context.TODO(),
        mongo.IndexModel{Keys: bson.D{{Key: "tenant_id", Value: 1}, {Key: "name", Value:
1}}, Options: options.Index().SetUnique(true)}}

    return nil
}

```

2) Config & Mongo client

```

// config/reco.yaml
mongo_uri: "mongodb://127.0.0.1:27017"
mongo_db: "reco"
server_addr: ":8088"
clock_skew_seconds: 300
pull:
  analytics_base_url: "https://analytics.example.com"
  page_size: 10000

// util/config.go
package util

import (
    "os"
    "gopkg.in/yaml.v3"
)

type PullCfg struct{ AnalyticsBaseURL string `yaml:"analytics_base_url"`; PageSize int
`yaml:"page_size"` }
type Cfg struct {
    MongoURI string `yaml:"mongo_uri"`
    MongoDB string `yaml:"mongo_db"`
    Server string `yaml:"server_addr"`
}

```

```

        ClockSkewSeconds int `yaml:"clock_skew_seconds"`
        Pull PullCfg `yaml:"pull"`
    }
    func LoadCfg(path string) (*Cfg, error) { b, e := os.ReadFile(path); if e != nil { return nil, e }; var c Cfg;
    return &c, yaml.Unmarshal(b,&c) }

// util/mongo.go
package util

import (
    "context"
    "time"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)
func MustMongo(uri string) *mongo.Client {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second); defer cancel()
    cli, err := mongo.Connect(ctx,
options.Client().ApplyURI(uri).SetMaxPoolSize(200).SetMinPoolSize(20))
    if err != nil { panic(err) }
    return cli
}

```

3) Security: HMAC verification + tenant secrets

```

// security/hmac.go
package security

import (
    "context"
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "errors"
    "strconv"
    "time"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"

```

```
)

var ErrSkew = errors.New("timestamp skew too large")
var ErrSig = errors.New("invalid signature")

// tenants: { tenant_id, name, hmac_secret: <Binary>, status: "active" }
func GetTenantSecret(ctx context.Context, db *mongo.Database, tenantID string) ([]byte, bool,
error) {
    var row struct {
        Status string `bson:"status"`
        HMACSecret []byte `bson:"hmac_secret"`
    }
    err := db.Collection("tenants").FindOne(ctx, bson.M{"tenant_id": tenantID}).Decode(&row)
    if err != nil { return nil, false, err }
    return row.HMACSecret, row.Status == "active", nil
}

func VerifyHMAC(body []byte, ts int64, hexSig string, skewSec int, secret []byte) error {
    if d := time.Now().Unix() - ts; d > int64(skewSec) || d < -int64(skewSec) { return ErrSkew }
    m := hmac.New(sha256.New, secret)
    m.Write([]byte(strconv.FormatInt(ts, 10)))
    m.Write([]byte("\n"))
    m.Write(body)
    exp := m.Sum(nil)
    got, err := hex.DecodeString(hexSig); if err != nil { return ErrSig }
    if !hmac.Equal(exp, got) { return ErrSig }
    return nil
}
```

4) Adapters (domain mapping)

```
// adapter/adapter.go
package adapter
type Raw = map[string]any

type Item struct {
    TenantID string
    ItemID string
}
```

```
    Type string
    Title string
    Desc string
    Facets map[string]any
    DurationSeconds *int
    Availability map[string]any
    Compliance map[string]any
}

type Interaction struct {
    TenantID string
    UserID string
    ItemID *string
    EventType string
    ValueInt *int
    PlaySecs *int
    DurationSecs *int
    TSISO string
    SessionID *string
    Context map[string]any
}

type DomainAdapter interface {
    Name() string
    MapItem(raw Raw) (Item, error)
    MapInteraction(raw Raw) (Interaction, error)
}

// adapter/ott.go (example)
package adapter
func ptr[T any](v T)*T{ return &v }

type OTT struct{}
func (o *OTT) Name() string { return "ott" }

func (o *OTT) MapItem(r Raw) (Item, error) {
    dur := 0; if v,ok := r["duration_seconds"].(int); ok { dur = v }
    return Item{
        TenantID: r["tenant_id"].(string),
        ItemID: r["item_id"].(string),
```



```

        Type: "content",
        Title: r["title"].(string),
        Desc: str(r,"desc"),
        Facets: map[string]any{"genre":r["genre"],"lang":r["lang"],"series_id":r["series_id"]},
        DurationSeconds: ifnz(dur),
        Availability: map[string]any{"geo":r["geo"],"device":r["device"]},
        Compliance: map[string]any{"age_18_plus": r["age_18_plus"]},
    }, nil
}

func (o *OTT) MapInteraction(r Raw) (Interaction, error) {
    var iid *string
    if v,ok := r["item_id"].(string); ok { iid=&v }
    var val *int; if v,ok := r["value"].(int); ok { val=&v }
    var ps *int; if v,ok := r["play_seconds"].(int); ok { ps=&v }
    var ds *int; if v,ok := r["duration_seconds"].(int); ok { ds=&v }

    return Interaction{
        TenantID: r["tenant_id"].(string),
        UserID: r["user_id"].(string),
        ItemID: iid,
        EventType: r["event_type"].(string),
        ValueInt: val,
        PlaySecs: ps,
        DurationSecs: ds,
        TSISO: r["ts"].(string),
        SessionID: strp(r,"session_id"),
        Context: map[string]any{"device":r["device"],"geo":r["geo"]},
    }, nil
}

func str(m Raw,k string)string{ if v,ok:=m[k].(string);ok{ return v }; return "" }
func strp(m Raw,k string)*string{ if v,ok:=m[k].(string);ok{ return &v }; return nil }
func ifnz(i int)*int{ if i==0 { return nil }; return &i }

```

5) Push ingestion handlers (HMAC + idempotency)

```

// handlers/ingest.go
package handlers

```

```

import (
    "context"

```

```

"encoding/json"
"io"
"net/http"
"strconv"
"time"

"go.mongodb.org/mongo-driver/bson"
"go.mongodb.org/mongo-driver/mongo"
"your/module/adapter"
"your/module/security"
)

type Ingest struct {
    DB      *mongo.Database
    Adapters map[string]adapter.DomainAdapter
    SkewSec int
}

func (h *Ingest) Routes(mux *http.ServeMux) {
    mux.HandleFunc("/reco/ingest/item", h.ingestItem)
    mux.HandleFunc("/reco/ingest/event", h.ingestEvent)
}

func (h *Ingest) ingestItem(w http.ResponseWriter, r *http.Request) {
    tenant := r.URL.Query().Get("tenant_id")
    adName := r.URL.Query().Get("adapter")
    ad, ok := h.Adapters[adName]
    if tenant=="" || !ok { http.Error(w, "bad tenant/adapter", 400); return }

    body, _ := io.ReadAll(r.Body)
    if err := h.verify(tenant, r.Header.Get("X-Timestamp"), r.Header.Get("X-Signature"), body);
err != nil {
        http.Error(w, err.Error(), 401); return
    }

    var raw map[string]any
    if err := json.Unmarshal(body, &raw); err != nil { http.Error(w, "bad json", 400); return }
    it, err := ad.MapItem(raw); if err != nil { http.Error(w, "map error", 400); return }

    doc := bson.M{

```

```

        "tenant_id": it.TenantID, "item_id": it.ItemID, "type": it.Type,
        "title": it.Title, "description": it.Desc,
        "facets": it.Facets, "duration_seconds": it.DurationSeconds,
        "availability": it.Availability, "compliance": it.Compliance,
        "updated_at": time.Now(),
    }
    _, err = h.DB.Collection("items").UpdateOne(context.TODO(),
        bson.M{"tenant_id": it.TenantID, "item_id": it.ItemID},
        bson.M{"$set": doc}, &mongo.UpdateOptions{Upsert: ptr(true)})
    if err != nil { http.Error(w, "db error", 500); return }
    w.Write([]byte(`{"ok":true}`))
}

func (h *Ingest) ingestEvent(w http.ResponseWriter, r *http.Request) {
    tenant := r.URL.Query().Get("tenant_id")
    adName := r.URL.Query().Get("adapter")
    idem := r.Header.Get("Idempotency-Key")
    ad, ok := h.Adapters[adName]
    if tenant=="" || !ok { http.Error(w, "bad tenant/adapter", 400); return }
    if idem=="" { http.Error(w, "missing Idempotency-Key", 400); return }

    body, _ := io.ReadAll(r.Body)
    if err := h.verify(tenant, r.Header.Get("X-Timestamp"), r.Header.Get("X-Signature"), body);
err != nil {
        http.Error(w, err.Error(), 401); return
    }

    // idempotency check
    _, err := h.DB.Collection("idempotency_keys").InsertOne(context.TODO(), bson.M{
        "tenant_id": tenant, "idem_key": idem, "created_at": time.Now(),
    })
    if mongo.IsDuplicateKeyError(err) { w.Write([]byte(`{"ok":true,"duplicate":true}`)); return }
    if err != nil { http.Error(w, "idem db error", 500); return }

    var raw map[string]any
    if err := json.Unmarshal(body, &raw); err != nil { http.Error(w, "bad json", 400); return }
    ev, err := ad.MapInteraction(raw); if err != nil { http.Error(w, "map error", 400); return }

    doc := bson.M{
        "tenant_id": ev.TenantID, "user_id": ev.UserID, "item_id": ev.ItemID,

```

```

        "event_type": ev.EventType, "value_int": ev.ValueInt,
        "play_seconds": ev.PlaySecs, "duration_seconds": ev.DurationSecs,
        "ts": ev.TSISO, "session_id": ev.SessionID, "context": ev.Context,
        "created_at": time.Now(),
    }
    _, err = h.DB.Collection("interactions").InsertOne(context.TODO(), doc)
    if err != nil { http.Error(w, "db error", 500); return }
    w.Write([]byte(`{"ok":true}`))
}

func (h *Ingest) verify(tenant, tsStr, sig string, body []byte) error {
    sec, active, err := security.GetTenantSecret(context.TODO(), h.DB, tenant)
    if err != nil || !active { return http.ErrNoCookie } // simple 401
    t, _ := strconv.ParseInt(tsStr, 10, 64)
    return security.VerifyHMAC(body, t, sig, h.SkewSec, sec)
}

func ptr[T any](v T) *T { return &v }

```

6) Optional Pull job (poll your Analytics API)

// cmd/pull/main.go

package main

```

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "your/module/util"
)

func main() {
    cfg, _ := util.LoadCfg("config/reco.yaml")
    cli := util.MustMongo(cfg.MongoURI)

```

```

db := cli.Database(cfg.MongoDB)
_ = util.EnsureIngestionIndexes(db)

tenant := "acme-ott"
name := "analytics_events"

// read watermark
var wm struct{ LastTS time.Time `bson:"last_ts"` }
_ = db.Collection("watermarks").FindOne(context.Background(),
    bson.M{"tenant_id":tenant,"name":name}).Decode(&wm)
since := wm.LastTS

for {
    url := fmt.Sprintf("%s/events?tenant_id=%s&since=%s&limit=%d",
        cfg.Pull.AnalyticsBaseURL, tenant,
        since.UTC().Format(time.RFC3339Nano), cfg.Pull.PageSize)

    resp, err := http.Get(url); if err != nil { panic(err) }
    b, _ := io.ReadAll(resp.Body); _ = resp.Body.Close()

    var events []map[string]any
    if err := json.Unmarshal(b, &events); err != nil { panic(err) }
    if len(events)==0 { fmt.Println("no more"); break }

    // upsert into interactions (skip HMAC here since we trust our own API)
    for _, e := range events {
        e["tenant_id"] = tenant
        // basic insert (reuse adapter if shapes differ)
        db.Collection("interactions").InsertOne(context.Background(), e)
        if ts, ok := e["ts"].(string); ok {
            if t, err := time.Parse(time.RFC3339Nano, ts); err==nil &&
t.After(since){ since = t }
        }
    }
    // advance watermark
    _, _ = db.Collection("watermarks").UpdateOne(context.Background(),
        bson.M{"tenant_id":tenant,"name":name},
        bson.M{"$set": bson.M{"last_ts": since}},
        options.Update().SetUpsert(true))
}

```

```

        if len(events) < cfg.Pull.PageSize { break }
    }
}

```

7) Server wiring (push endpoints)

```

// cmd/server/main.go
package main

import (
    "net/http"
    "your/module/adapter"
    "your/module/handlers"
    "your/module/util"
)

func main() {
    cfg, _ := util.LoadCfg("config/reco.yaml")
    cli := util.MustMongo(cfg.MongoURI)
    db := cli.Database(cfg.MongoDB)
    _ = util.EnsureIngestionIndexes(db)

    mux := http.NewServeMux()
    ing := &handlers.Ingest{
        DB: db,
        Adapters: map[string]adapter.DomainAdapter{
            "ott": &adapter.OTT{},
            // "pharma": &adapter.Pharma{}, "consulting": &adapter.Consulting{} ...
        },
        SkewSec: cfg.ClockSkewSeconds,
    }
    ing.Routes(mux)

    http.ListenAndServe(cfg.Server, mux)
}

```

8) Example request (Push mode)

Headers

X-Timestamp: 1724056802

X-Signature: <hex hmac sha256 over "timestamp\n<body>">

Idempotency-Key: evt-abc-123

Content-Type: application/json

Body (event)

```
{
  "tenant_id":"acme-ott",
  "user_id":"U123",
  "item_id":"C456",
  "event_type":"play",
  "value":1,
  "play_seconds":137,
  "duration_seconds":1800,
  "ts":"2025-08-19T08:30:02Z",
  "session_id":"S-abc",
  "device":"android",
  "geo":"IN"
}
```

POST /reco/ingest/item?tenant_id=T&adapter=A

POST /reco/ingest/event?tenant_id=T&adapter=A

Phase 3 – Precompute jobs (trending / co-visit / content-sim).

Online service that pulls precompute features + user history, blends them with scoring weights, applies MMR diversity and policies (age, geo, dedupe).

Caches results in Azure Cache for Redis for low latency and returns a ranked recommendation list as JSON.

Shared helpers

lib/reco/common.go

package reco

```
import (  
    "sort"  
    "time"  
)
```

```
type ItemScore struct {  
    CID string  
    Score float64  
}
```

```
func TopK(items []ItemScore, k int) []ItemScore {  
    if len(items) <= k { return items }  
    sort.Slice(items, func(i, j int) bool { return items[i].Score > items[j].Score })  
    return items[:k]  
}
```

```
func DayInt(t time.Time) int { return t.Year()*10000 + int(t.Month()*100 + t.Day()) }
```

1) Trending (7-day, exponential recency + completion)

- Score per event: $\exp(-\alpha * \text{age_hours}) * \min(1, \text{play_seconds}/\text{duration_seconds})$
- Sum by item; optional unique-user multiplier via log.

lib/reco/precompute_trending.go

package reco

```
import (  
    "context"  
    "time"
```



```

"go.mongodb.org/mongo-driver/bson"
"go.mongodb.org/mongo-driver/mongo"
"go.mongodb.org/mongo-driver/mongo/options"
)

type TrendingConfig struct {
    LookbackDays int // e.g., 7
    DecayPerHour float64 // e.g., 0.15
    Limit int // e.g., 1000
    UseUniqueUserBoost bool // multiply by log(2 + unique_users)
}

func RunTrending(ctx context.Context, db *mongo.Database, tenantID string, cfg TrendingConfig,
now time.Time) error {
    from := now.Add(-time.Duration(cfg.LookbackDays) * 24 * time.Hour)

    pipe := mongo.Pipeline{
        {{Key: "$match", Value: bson.M{
            "tenant_id": tenantID,
            "ts": bson.M{"$gte": from},
        }},
        {{Key: "$addFields", Value: bson.M{
            "ageH": bson.M{"$divide": []any{
                bson.M{"$subtract": []any{now, "$ts"}}, float64(time.Hour /
time.Millisecond),
            }},
            "comp": bson.M{"$min": []any{1, bson.M{"$divide": []any{
                "$duration_seconds",
                "$duration_seconds", // will be replaced by max(1,duration) below
            }},
        }},
        // Fix comp = min(1, play_seconds/max(1,duration_seconds))
        {{Key: "$addFields", Value: bson.M{
            "comp": bson.M{"$min": []any{1, bson.M{"$divide": []any{
                "$play_seconds", bson.M{"$max": []any{"$duration_seconds", 1}},
            }},
        }},
        {{Key: "$project", Value: bson.M{
            "tenant_id": 1, "c_id": 1,

```

```

        "s": bson.M{"$multiply": []any{
            "$comp",
            bson.M{"$exp": bson.M{"$multiply": []any{-cfg.DecayPerHour,
"$ageH"}}},
        }},
        "u_id": 1,
    }},
    {{Key: "$group", Value: bson.M{
        "_id": "$c_id",
        "score": bson.M{"$sum": "$s"},
        "users": bson.M{"$addToSet": "$u_id"},
    }},
    }
    if cfg.UseUniqueUserBoost {
        pipe = append(pipe,
            bson.D{{Key: "$addFields", Value: bson.M{
                "uuc": bson.M{"$size": "$users"},
                "score": bson.M{"$multiply": []any{"$score", bson.M{"$log":
bson.M{"$add": []any{2, "$uuc"}}}},
            }},
            bson.D{{Key: "$project", Value: bson.M{"users": 0}}},
        )
    } else {
        pipe = append(pipe, bson.D{{Key: "$project", Value: bson.M{"users": 0, "score": 1}}})
    }

    pipe = append(pipe,
        bson.D{{Key: "$sort", Value: bson.M{"score": -1}}},
        bson.D{{Key: "$limit", Value: cfg.Limit}},
    )

    cur, err := db.Collection("interactions").Aggregate(ctx, pipe, &options.AggregateOptions{
        AllowDiskUse: ptr(true),
    })
    if err != nil { return err }
    defer cur.Close(ctx)

    var list []ItemScore
    for cur.Next(ctx) {
        var row struct {

```

```

                ID string `bson:"_id"`
                Score float64 `bson:"score"`
            }
            if err := cur.Decode(&row); err != nil { return err }
            list = append(list, ItemScore{CID: row.ID, Score: row.Score})
        }

        day := DayInt(now)
        doc := bson.M{"tenant_id": tenantID, "day": day, "items": list, "updated_at": now}
        _, err = db.Collection("popular_day").UpdateOne(ctx,
            bson.M{"tenant_id": tenantID, "day": day},
            bson.M{"$set": doc},
            options.Update().SetUpsert(true),
        )
        return err
    }
}

func ptr[T any](v T) *T { return &v }

```

CLI entrypoint

cmd/precompute/trending/main.go

package main

```

import (
    "context"
    "log"
    "time"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "your/module/lib/reco"
)

func main() {
    mcli, _ := mongo.Connect(context.Background(),
options.Client().ApplyURI("mongodb://127.0.0.1:27017"))

```

```

db := mcli.Database("reco")

err := reco.RunTrending(context.Background(), db, "acme-ott", reco.TrendingConfig{
    LookbackDays: 7, DecayPerHour: 0.15, Limit: 1000, UseUniqueUserBoost: true,
}, time.Now())
if err != nil { log.Fatal(err) }
log.Println("trending done")
}

```

2) Co-visitation (item → item neighbors)

- Sessionize by (tenant_id, user_id) with a gap ≤ 45 min.
- For each session, for ordered items, add pairs (A→B and B→A) with weight $1/(1+\Delta t_{\text{minutes}})$.
- Aggregate and keep Top-K per source.

lib/reco/precompute_covisit.go

```
package reco
```

```

import (
    "context"
    "math"
    "sort"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

```

```

type CoVisitConfig struct {
    LookbackDays int // e.g., 14 or 30
    SessionGapMin int // e.g., 45
    TopKPerItem int // e.g., 100
}

```

```

    MaxUserRows int64 // safety cap per tenant batch (0 = no cap)
}

type playRow struct {
    UID string `bson:"u_id"`
    CID string `bson:"c_id"`
    TS time.Time `bson:"ts"`
}

func RunCoVisit(ctx context.Context, db *mongo.Database, tenantID string, cfg CoVisitConfig,
now time.Time) error {
    from := now.Add(-time.Duration(cfg.LookbackDays) * 24 * time.Hour)

    findFilter := bson.M{"tenant_id": tenantID, "ts": bson.M{"$gte": from}}
    opts :=
options.Find().SetProjection(bson.M{"u_id":1,"c_id":1,"ts":1,"_id":0}).SetSort(bson.M{"u_id":1,"ts":1}
)

    if cfg.MaxUserRows > 0 { opts.SetLimit(cfg.MaxUserRows) }

    cur, err := db.Collection("interactions").Find(ctx, findFilter, opts)
    if err != nil { return err }
    defer cur.Close(ctx)

    // 1) Group plays by user
    sessions := make(map[string][]playRow, 1<<15)
    for cur.Next(ctx) {
        var p playRow
        if err := cur.Decode(&p); err != nil { return err }
        sessions[p.UID] = append(sessions[p.UID], p)
    }

    // 2) Build weighted pairs
    type key struct{ A, B string }
    weights := make(map[key]float64, 1<<20)

    for _, seq := range sessions {
        // seq is already sorted by ts due to query sort
        n := len(seq)
        for i := 0; i < n; i++ {
            for j := i + 1; j < n; j++ {

```

```

        dtMin := seq[j].TS.Sub(seq[i].TS).Minutes()
        if dtMin > float64(cfg.SessionGapMin) { break }
        w := 1.0 / (1.0 + dtMin)
        if math.IsInf(w, 0) || math.IsNaN(w) { continue }
        weights[key{seq[i].CID, seq[j].CID}] += w
        weights[key{seq[j].CID, seq[i].CID}] += w
    }
}

// 3) Group by src and keep TopK
bySrc := map[string][]ItemScore{}
for k, sc := range weights {
    bySrc[k.A] = append(bySrc[k.A], ItemScore{CID: k.B, Score: sc})
}
bulk := []mongo.WriteModel{}
for src, sims := range bySrc {
    sort.Slice(sims, func(i, j int) bool { return sims[i].Score > sims[j].Score })
    if len(sims) > cfg.TopKPerItem { sims = sims[:cfg.TopKPerItem] }

    doc := bson.M{"tenant_id": tenantID, "src": src, "sims": sims, "updated_at": now}
    bulk = append(bulk, mongo.NewReplaceOneModel().
        SetFilter(bson.M{"tenant_id": tenantID, "src": src}).
        SetReplacement(doc).
        SetUpsert(true),
    )
    // flush in chunks to avoid huge batches
    if len(bulk) >= 1000 {
        if _, err := db.Collection("covisit_topk").BulkWrite(ctx, bulk,
options.BulkWrite().SetOrdered(false)); err != nil {
            return err
        }
        bulk = bulk[:0]
    }
}
if len(bulk) > 0 {
    if _, err := db.Collection("covisit_topk").BulkWrite(ctx, bulk,
options.BulkWrite().SetOrdered(false)); err != nil {
        return err
    }
}

```

```
    }
    return nil
}
```

CLI

cmd/precompute/covisit/main.go

```
package main
```

```
import (
    "context"
    "log"
    "time"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "your/module/lib/reco"
)

func main() {
    mcli, _ := mongo.Connect(context.Background(),
options.Client().ApplyURI("mongodb://127.0.0.1:27017"))
    db := mcli.Database("reco")

    err := reco.RunCoVisit(context.Background(), db, "acme-ott", reco.CoVisitConfig{
        LookbackDays: 14, SessionGapMin: 45, TopKPerItem: 100, MaxUserRows: 0,
    }, time.Now())
    if err != nil { log.Fatal(err) }
    log.Println("covisit done")
}
```

3) Content-Similarity (metadata facets)

- Use tenant-specific facet weights (e.g., OTT: `genre=1.0, series_id=0.7, lang=0.3`; Consulting: `industry=1.0, practice=0.8 ...`).
- Compute cosine on binary facet vectors (or Jaccard). Below is cosine on weighted binary features.

```
lib/reco/precompute_contentsim.go
```

```
package reco
```

```
import (
    "context"
    "math"
    "sort"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)
```

```
// Example: map facet->weight; load from tenant YAML in real use
type FacetWeights map[string]float64
```

```
type ContentSimConfig struct {
    TopKPerItem int
    FacetWeights FacetWeights // e.g., {"genre":1.0, "series_id":0.7, "lang":0.3}
}
```

```
type itemDoc struct {
    ItemID string      `bson:"item_id"`
    Facets map[string]any  `bson:"facets"`
}
```

```
func RunContentSim(ctx context.Context, db *mongo.Database, tenantID string, cfg
ContentSimConfig, now time.Time) error {
    cur, err := db.Collection("items").Find(ctx,
        bson.M{"tenant_id": tenantID},
        options.Find().SetProjection(bson.M{"item_id":1,"facets":1,"_id":0})),
```



```

)
if err != nil { return err }
defer cur.Close(ctx)

var items []itemDoc
for cur.Next(ctx) {
    var it itemDoc
    if err := cur.Decode(&it); err != nil { return err }
    items = append(items, it)
}

// Precompute vector norms
type vec struct {
    id string
    wmap map[string]float64 // key = facet=value (joined)
    norm float64
}
vecs := make([]vec, 0, len(items))
for _, it := range items {
    wm := map[string]float64{}
    for fk, wt := range cfg.FacetWeights {
        // facet can be string or array; handle both
        if val, ok := it.Facets[fk]; ok && wt > 0 {
            switch v := val.(type) {
            case string:
                wm[fk+"="+v] = wt
            case []any:
                for _, x := range v {
                    if s, ok := x.(string); ok {
                        wm[fk+"="+s] = wt
                    }
                }
            }
        }
    }

    var sum float64
    for _, w := range wm { sum += w * w }
    vecs = append(vecs, vec{id: it.ItemID, wmap: wm, norm: math.Sqrt(sum)})
}

```

```

// Pairwise similarities (O(N^2) over catalog; fine for ~5k-50k items; for bigger, shard by
facet)
type pair struct{ a, b string }
scores := map[pair]float64{}
for i := 0; i < len(vecs); i++ {
    for j := i + 1; j < len(vecs); j++ {
        dot := 0.0
        for k, w := range vecs[i].wmap {
            if w2, ok := vecs[j].wmap[k]; ok {
                dot += w * w2
            }
        }
        if dot == 0 || vecs[i].norm == 0 || vecs[j].norm == 0 { continue }
        cos := dot / (vecs[i].norm * vecs[j].norm)
        if cos <= 0 { continue }
        scores[pair{vecs[i].id, vecs[j].id}] = cos
        scores[pair{vecs[j].id, vecs[i].id}] = cos
    }
}

// Group per src and keep TopK
bySrc := map[string][]ItemScore{}
for p, sc := range scores {
    bySrc[p.a] = append(bySrc[p.a], ItemScore{CID: p.b, Score: sc})
}
bulk := []mongo.WriteModel{}
for src, sims := range bySrc {
    sort.Slice(sims, func(i, j int) bool { return sims[i].Score > sims[j].Score })
    if len(sims) > cfg.TopKPerItem { sims = sims[:cfg.TopKPerItem] }

    doc := bson.M{"tenant_id": tenantID, "src": src, "sims": sims, "updated_at": now}
    bulk = append(bulk, mongo.NewReplaceOneModel().
        SetFilter(bson.M{"tenant_id": tenantID, "src": src}).
        SetReplacement(doc).
        SetUpsert(true),
    )
    if len(bulk) >= 1000 {
        if _, err := db.Collection("content_sim_topk").BulkWrite(ctx, bulk,
options.BulkWrite().SetOrdered(false)); err != nil {
            return err
        }
    }
}

```

```

        }
        bulk = bulk[:0]
    }
}
if len(bulk) > 0 {
    if _, err := db.Collection("content_sim_topk").BulkWrite(ctx, bulk,
options.BulkWrite().SetOrdered(false)); err != nil {
        return err
    }
}
return nil
}

```

CLI

cmd/precompute/contentsim/main.go

package main

```

import (
    "context"
    "log"
    "time"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "your/module/lib/reco"
)

```

```

func main() {
    mcli, _ := mongo.Connect(context.Background(),
options.Client().ApplyURI("mongodb://127.0.0.1:27017"))
    db := mcli.Database("reco")

    err := reco.RunContentSim(context.Background(), db, "acme-ott", reco.ContentSimConfig{
        TopKPerItem: 50,
        FacetWeights: reco.FacetWeights{
            "genre": 1.0, "series_id": 0.7, "lang": 0.3,

```

```

        // Pharma/Consulting tenants will use different facet weights
    },
    }, time.Now())
    if err != nil { log.Fatal(err) }
    log.Println("content-sim done")
}

```

2) User Recommendations API (explicit `user_id`)

Endpoint

GET `/v1/users/{user_id}/recommendations?tenant_id=T&app_id=A&context=home&limit=30`

Example

```

curl
"http://localhost:8080/v1/users/U123/recommendations?tenant_id=acme-ott&app_id=web&limit=20"

```

Response

```

{
  "user_id": "U123",
  "items": [
    {"c_id": "I902", "score": 1.92, "why": {"covisit": 0.8, "content": 0.5, "trending": 0.3}},
    {"c_id": "I441", "score": 1.75, "why": {"covisit": 0.6, "trending": 0.7}}
  ]
}

```

3) Handler snippet (pulls seeds by `u_id`, suggests content)

```

// api/user_reco.go
func (h *RecoHandler) UserRecommendations(w http.ResponseWriter, r *http.Request) {
    parts := strings.Split(strings.Trim(r.URL.Path, "/"), "/")

```

```

if len(parts) < 4 || parts[0]!="v1" || parts[1]!="users" || parts[3]!="recommendations" {
    http.NotFound(w,r); return
}
uid := parts[2]                // <-- user_id from URL
tenant := r.URL.Query().Get("tenant_id")
app := r.URL.Query().Get("app_id")
k,_ := strconv.Atoi(r.URL.Query().Get("limit")); if k==0 { k = h.Cfg.LimitDefault }

// Per-user cache key
cacheKey := fmt.Sprintf("reco:%s:%s:%s:%s:%d", tenant, app, uid, "home", k)
if bs,_ := h.RDB.Get(r.Context(), cacheKey).Bytes(); len(bs) > 0 {
    w.Header().Set("Content-Type","application/json"); w.Write(bs); return
}

st := reco.Stores{DB: h.DB}
// 1) Seeds from this user's history (uses u_id in Mongo)
seeds := recentUserItems(r.Context(), h.DB, tenant, app, uid, h.Cfg.SeedsFromUserHistory)

// 2) Build candidates from co-visit, content-sim, trending (+ optional user_topN)
feat := map[string]reco.Feat{}
for _, s := range seeds {
    mergeMax(feat, st.Neighbors(r.Context(), tenant, app, s, h.Cfg.PoolCovisitPerSeed),
        func(f *reco.Feat, v float64){ f.Covisit = max(f.Covisit, v) })
    mergeMax(feat, st.ContentSims(r.Context(), tenant, app, s, h.Cfg.PoolContentPerSeed),
        func(f *reco.Feat, v float64){ f.Content = max(f.Content, v) })
}
// (Optional) Per-user CF candidates (ALS)
mergeMax(feat, st.FetchUserTopN(r.Context(), tenant, app, uid, 200),
    func(f *reco.Feat, v float64){ f.Covisit = max(f.Covisit, v) })

day := reco.DayInt(time.Now())
mergeMax(feat, st.Trending(r.Context(), tenant, app, day, h.Cfg.PoolTrending),
    func(f *reco.Feat, v float64){ f.Trending = max(f.Trending, v) })

// 3) Penalize recently seen items for this user
seen := recentUserSet(r.Context(), h.DB, tenant, app, uid, 72)
for cid := range feat { if seen[cid] { feat[cid].SeenPenalty = 1 } }

// 4) Score, diversify, apply policies
scored := rankFromFeat(feat, h.Cfg.Weights)

```

```

sim := func(a,b string) float64 { sims := st.ContentSims(r.Context(), tenant, app, a, 50); for _,x :=
range sims { if x.CID==b { return x.Score } }; return 0 }
diversified := reco.MMR(scored, sim, h.Cfg.MMRLambda, k*2)

pol := reco.PolicyInput{ Seen: seen } // extend with lang/geo/HCP/NDA per tenant/app
out := make([]reco.Scored,0,k)
for _, it := range diversified {
    meta := fetchItemMeta(r.Context(), h.DB, tenant, app, it.CID)
    if reco.Allow(meta, pol) { out = append(out, it); if len(out)==k { break } }
}

resp := struct{ UserID string `json:"user_id"`; Items []reco.Scored `json:"items"` }{uid, out}
bs, _ := json.Marshal(resp)
_ = h.RDB.Set(r.Context(), cacheKey, bs, h.Cfg.CacheTTL).Err()
w.Header().Set("Content-Type","application/json"); w.Write(bs)
}

```

Helpers (make sure they use `u_id`)

```

func recentUserItems(ctx context.Context, db *mongo.Database, tenant, app, uid string, n int)
[]string {
    cur, _ := db.Collection("interactions").Find(ctx,
        bson.M{"tenant_id":tenant,"app_id":app,"u_id":uid,"c_id":bson.M{"$ne":nil},
            "event_type": bson.M{"$in":[]string{"play","view"}}},
        options.Find().SetProjection(bson.M{"c_id":1,"_id":0}).SetSort(bson.M{"ts":-1}).SetLimit(int64(n*3))
    ,
    )
    defer cur.Close(ctx)
    // dedupe by c_id, keep recency
    m := map[string]struct{}{}; out := []string{}
    for cur.Next(ctx) {
        var r struct{ CID *string `bson:"c_id"` }
        _ = cur.Decode(&r); if r.CID==nil { continue }
        if _,seen := m[*r.CID]; seen { continue }
        m[*r.CID]=struct{}{}; out = append(out, *r.CID)
        if len(out) >= n { break }
    }
    return out
}

```

```

}

func recentUserSet(ctx context.Context, db *mongo.Database, tenant, app, uid string, hours int)
map[string]bool {
    from := time.Now().Add(-time.Duration(hours)*time.Hour)
    cur, _ := db.Collection("interactions").Find(ctx,

bson.M{"tenant_id":tenant,"app_id":app,"u_id":uid,"ts":bson.M{"$gte":from},"c_id":bson.M{"$ne":nil
}},
    options.Find().SetProjection(bson.M{"c_id":1,"_id":0}).SetSort(bson.M{"ts":-1}).SetLimit(500),
)
    defer cur.Close(ctx)
    m := map[string]bool{}
    for cur.Next(ctx) {
        var r struct{ CID *string `bson:"c_id"` }
        _ = cur.Decode(&r); if r.CID!=nil { m[*r.CID]=true }
    }
    return m
}

```

4) Suggested cron (system)

```

# Trending every 30 min
*/30 * * * * /usr/local/bin/go run ./cmd/precompute/trending/main.go >> /var/log/reco_trending.log
2>&1

# Co-visit every 6 hours
0 */6 * * * /usr/local/bin/go run ./cmd/precompute/covisit/main.go >> /var/log/reco_covisit.log
2>&1

# Content-sim every 12 hours (or on catalog change)
0 */12 * * * /usr/local/bin/go run ./cmd/precompute/contentsim/main.go >> /var/log/reco_csim.log
2>&1

```

5) Notes & scaling tips

- Ensure these indexes exist (once):
 - interactions: {tenant_id:1, ts:-1}, {tenant_id:1, u_id:1, ts:-1}, {tenant_id:1, item_id:1, ts:-1}
 - items: {tenant_id:1, item_id:1} unique
 - covisit_topk: {tenant_id:1, src:1}, content_sim_topk: {tenant_id:1, src:1}, popular_day: {tenant_id:1, day:1}
- For very large catalogs (>50k items), switch content-sim to blocking by facet (e.g., only compare items sharing **genre** or **industry**) to avoid $O(N^2)$.
- **RunCoVisit** reads all interactions in the window; if huge, shard by date or user ranges and run multiple workers.

Phase 3 – Online serving (low latency) (1–2 days)

0) Config (add Azure Redis + knobs)

config/reco.yaml


```
mongo_uri: "mongodb://127.0.0.1:27017"
mongo_db: "reco"
```

```
# Azure Cache for Redis (TLS @ 6380)
redis_addr: "<your-cache-name>.redis.cache.windows.net:6380"
redis_password: "<primary_or_secondary_key>"
redis_db: 0
cache_ttl_seconds: 600
```

```
scoring:
  weights: { covisit: 0.45, trending: 0.30, content: 0.15, fresh_bonus: 0.10, seen_penalty: 0.10 }
  mmr_lambda: 0.7
limits:
  recommend_limit_default: 30
  pool_covisit_per_seed: 50
  pool_trending: 200
  pool_content_per_seed: 50
  seeds_from_user_history: 5
  freshness_hours: 168    # 7 days
```

1) Azure Redis client (TLS)

```
// util/redis_azure.go
package util

import (
    "context"
    "crypto/tls"
    "time"

    "github.com/redis/go-redis/v9"
)

var rdb *redis.Client

type RedisCfg struct {
    Addr    string
    Password string
```

```

    DB    int
    TTL   time.Duration
}

func NewAzureRedis(cfg RedisCfg) *redis.Client {
    if rdb != nil { return rdb }
    rdb = redis.NewClient(&redis.Options{
        Addr:    cfg.Addr, // "<name>.redis.cache.windows.net:6380"
        Password: cfg.Password, // access key
        DB:      cfg.DB,
        TLSConfig: &tls.Config{}, // TLS required on 6380
        PoolSize: 200, MinIdleConns: 20,
        DialTimeout: 2*time.Second, ReadTimeout: 20*time.Millisecond, WriteTimeout:
20*time.Millisecond,
    })
    _ = rdb.Ping(context.Background()).Err()
    return rdb
}

```

2) Scoring + MMR + Policies (tiny)

```

// lib/reco/score.go
package reco
type Weights struct{ Covisit, Trending, Content, FreshBonus, SeenPenalty float64 }
type Feat struct{ Covisit, Trending, Content, FreshnessBonus, SeenPenalty float64 }
type Scored struct{ CID string; Score float64; Why Feat }
func Score(f Feat, w Weights) float64 {
    return w.Covisit*f.Covisit + w.Trending*f.Trending + w.Content*f.Content +
        w.FreshBonus*f.FreshnessBonus - w.SeenPenalty*f.SeenPenalty
}

// lib/reco/mmr.go
package reco
import "math"
type SimFunc func(a,b string) float64
func MMR(items []Scored, sim SimFunc, lambda float64, k int) []Scored {
    if k > len(items) { k = len(items) }
    sel := make([]Scored,0,k); pool := append([]Scored(nil), items...)

```

```

    for len(sel)<k && len(pool)>0 {
        best, idx := -math.MaxFloat64, 0
        for i, it := range pool {
            maxSim := 0.0
            for _, s := range sel { if v:=sim(it.CID, s.CID); v>maxSim { maxSim=v } }
            val := lambda*it.Score - (1-lambda)*maxSim
            if val>best { best, idx = val, i }
        }
        sel = append(sel, pool[idx])
        pool = append(pool[:idx], pool[idx+1:]...)
    }
    return sel
}

// lib/reco/policy.go
package reco
type ItemMeta struct { CID string; Age18Plus bool; Lang string; Geo []string; SeriesID string }
type PolicyInput struct{ UserLang, UserGeo string; Seen map[string]bool }
func Allow(m ItemMeta, p PolicyInput) bool {
    if p.Seen[m.CID] { return false } // simple "don't repeat recently"
    // extend with geo/age/HCP/etc per tenant
    return true
}

```

3) Precompute readers + feature builders

```

// lib/reco/features.go
package reco

import (
    "context"
    "sort"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
)

```

```

type ItemScore struct { CID string `bson:"c_id"`; Score float64 `bson:"score"` }
type simsDoc struct { Src string `bson:"src"`; Sims []ItemScore `bson:"sims"` }
type topDoc struct { Items []ItemScore `bson:"items"` }

type Stores struct{ DB *mongo.Database }

func DayInt(t time.Time) int { return t.Year()*10000 + int(t.Month()*100 + t.Day() ) }

func (s Stores) Trending(ctx context.Context, tenant string, day int, limit int) []ItemScore {
    var row topDoc
    _ = s.DB.Collection("popular_day").FindOne(ctx, bson.M{"tenant_id":tenant,
"day":day}).Decode(&row)
    if len(row.Items)>limit { row.Items=row.Items[:limit] }
    return row.Items
}

func (s Stores) Neighbors(ctx context.Context, tenant, src string, limit int) []ItemScore {
    var d simsDoc; _ = s.DB.Collection("covisit_topk").FindOne(ctx,
bson.M{"tenant_id":tenant,"src":src}).Decode(&d)
    if len(d.Sims)>limit { d.Sims = d.Sims[:limit] }
    return d.Sims
}

func (s Stores) ContentSims(ctx context.Context, tenant, src string, limit int) []ItemScore {
    var d simsDoc; _ = s.DB.Collection("content_sim_topk").FindOne(ctx,
bson.M{"tenant_id":tenant,"src":src}).Decode(&d)
    if len(d.Sims)>limit { d.Sims = d.Sims[:limit] }
    return d.Sims
}

func mergeMax(dst map[string]Feat, src []ItemScore, add func(*Feat, float64)) {
    for _, it := range src {
        f := dst[it.CID]
        add(&f, it.Score)
        dst[it.CID] = f
    }
}

func rankFromFeat(m map[string]Feat, w Weights) []Scored {
    out := make([]Scored,0,len(m))
    for cid, f := range m { out = append(out, Scored{CID: cid, Score: Score(f,w), Why: f}) }
    sort.Slice(out, func(i,j int) bool { return out[i].Score > out[j].Score })
    return out
}

```

```
}
```

4) Recommend handler (caching + policies)

```
// api/recommend_handler.go
package api

import (
    "context"
    "encoding/json"
    "fmt"
    "net/http"
    "strconv"
    "time"

    "github.com/redis/go-redis/v9"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"

    "your/module/lib/reco"
)

type Config struct {
    Weights      reco.Weights
    MMRLambda    float64
    LimitDefault int
    PoolCovisitPerSeed int
    PoolTrending int
    PoolContentPerSeed int
    SeedsFromUserHistory int
    FreshnessHours int
    CacheTTL     time.Duration
}

type RecoHandler struct {
    DB *mongo.Database
    RDB *redis.Client
    Cfg Config
}
```

```

}

func (h *RecoHandler) Routes(mux *http.ServeMux) {
    mux.HandleFunc("/v1/recommend", h.Recommend)
}

func (h *RecoHandler) Recommend(w http.ResponseWriter, r *http.Request) {
    ctx := r.Context()
    tenant := r.URL.Query().Get("tenant_id")
    uid := r.URL.Query().Get("user_id")
    k, _ := strconv.Atoi(r.URL.Query().Get("limit"))
    if k==0 { k = h.Cfg.LimitDefault }

    cacheKey := fmt.Sprintf("reco:%s:%s:%d", tenant, uid, k)
    if bs, err := h.RDB.Get(ctx, cacheKey).Bytes(); err==nil && len(bs)>0 {
        w.Header().Set("Content-Type", "application/json"); w.Write(bs); return
    }

    st := reco.Stores{DB: h.DB}
    now := time.Now()
    day := reco.DayInt(now)

    // 1) Seeds from user history (last N items)
    seeds := recentUserItems(ctx, h.DB, tenant, uid, h.Cfg.SeedsFromUserHistory)

    // 2) Build candidate feature map
    feat := map[string]reco.Feat{}
    // 2a) neighbors from seeds (co-visit)
    for _, s := range seeds {
        ns := st.Neighbors(ctx, tenant, s, h.Cfg.PoolCovisitPerSeed)
        mergeMax(feat, ns, func(f *reco.Feat, v float64){ f.Covisit = max(f.Covisit, v) })
        // content-sim from seeds
        cs := st.ContentSims(ctx, tenant, s, h.Cfg.PoolContentPerSeed)
        mergeMax(feat, cs, func(f *reco.Feat, v float64){ f.Content = max(f.Content, v) })
    }
    // 2b) trending
    tr := st.Trending(ctx, tenant, day, h.Cfg.PoolTrending)
    mergeMax(feat, tr, func(f *reco.Feat, v float64){ f.Trending = max(f.Trending, v) })

    // 3) Freshness & seen penalties

```

```

seen := recentUserSet(ctx, h.DB, tenant, uid, 72) // seen in last 72h
for cid := range feat {
    if recentlyInTrending(tr, cid) && h.Cfg.FreshnessHours>0 {
feat[cid].FreshnessBonus = 1 }
    if seen[cid] { feat[cid].SeenPenalty = 1 }
}

// 4) Score
scored := rankFromFeat(feat, h.Cfg.Weights)

// 5) Diversity (MMR) using content-sim similarity as sim(a,b)
sim := func(a,b string) float64 {
    // cheap: look b in content_sim_topk(a)
    sims := st.ContentSims(ctx, tenant, a, 50)
    for _, x := range sims { if x.CID==b { return x.Score } }
    return 0
}
div := reco.MMR(scored, sim, h.Cfg.MMRLambda, k*2) // select a bit more before policy

// 6) Policies (example: drop recently seen)
pol := reco.PolicyInput{ UserLang:"", UserGeo:"", Seen: seen }
final := make([]reco.Scored,0,k)
for _, it := range div {
    meta := fetchItemMeta(ctx, h.DB, tenant, it.CID)
    if reco.Allow(meta, pol) {
        final = append(final, it)
        if len(final)==k { break }
    }
}

// 7) Response + cache
resp := struct{
    UserID string    `json:"user_id"`
    Items []reco.Scored `json:"items"`
}{uid, final}

bs, _ := json.Marshal(resp)
_ = h.RDB.Set(ctx, cacheKey, bs, h.Cfg.CacheTTL).Err()

w.Header().Set("Content-Type","application/json")

```

```

        w.Write(bs)
    }

func recentUserItems(ctx context.Context, db *mongo.Database, tenant, uid string, n int) []string {
    cur, _ := db.Collection("interactions").Find(ctx,
        bson.M{"tenant_id":tenant,"u_id":uid,"c_id": bson.M{"$ne": nil}},
        nil,
    )
    defer cur.Close(ctx)
    // you can add sort/limit projection; keeping brief:
    out := []string{}
    for cur.Next(ctx) {
        var row struct{ CID *string `bson:"c_id"` }
        _ = cur.Decode(&row); if row.CID!=nil { out = append(out, *row.CID) }
        if len(out)>=n { break }
    }
    return out
}

func recentUserSet(ctx context.Context, db *mongo.Database, tenant, uid string, hours int)
map[string]bool {
    from := time.Now().Add(-time.Duration(hours)*time.Hour)
    cur, _ := db.Collection("interactions").Find(ctx,

bson.M{"tenant_id":tenant,"u_id":uid,"ts":bson.M{"$gte":from},"c_id":bson.M{"$ne":nil}},
        nil,
    )
    defer cur.Close(ctx)
    m := map[string]bool{}
    for cur.Next(ctx) {
        var r struct{ CID *string `bson:"c_id"` }
        _ = cur.Decode(&r); if r.CID!=nil { m[*r.CID]=true }
    }
    return m
}

func recentlyInTrending(tr []reco.ItemScore, cid string) bool {
    for _, x := range tr { if x.CID==cid { return true } }
    return false
}

```



```
func fetchItemMeta(ctx context.Context, db *mongo.Database, tenant, cid string) reco.ItemMeta {
    var d struct{
        Age bool `bson:"compliance.age_18_plus"`
        Lg string `bson:"facets.lang"`
        Geo []string `bson:"availability.geo"`
        Series string `bson:"facets.series_id"`
    }
    _ = db.Collection("items").FindOne(ctx,
bson.M{"tenant_id":tenant,"item_id":cid}).Decode(&d)
    return reco.ItemMeta{CID:cid, Age18Plus:d.Age, Lang:d.Lg, Geo:d.Geo, SeriesID:d.Series}
}

func max(a,b float64) float64 { if b>a { return b }; return a }
```

5) Wire it up (main)

```
// cmd/service/main.go
package main

import (
    "net/http"
    "time"

    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

    "your/module/api"
    "your/module/lib/reco"
    "your/module/util"
)

func main() {
    // load your YAML here (omitted for brevity). Hardcoding a bit:
    redis := util.NewAzureRedis(util.RedisCfg{
        Addr: "<name>.redis.cache.windows.net:6380",
        Password: "<key>",
        DB: 0, TTL: 10*time.Minute,
```

```

})
mcli,_:= mongo.Connect(nil, options.Client().ApplyURI("mongodb://127.0.0.1:27017"))
db:= mcli.Database("reco")

h:= &api.RecoHandler{
    DB: db,
    RDB: redis,
    Cfg: api.Config{
        Weights: reco.Weights{Covisit:0.45, Trending:0.30, Content:0.15,
FreshBonus:0.10, SeenPenalty:0.10},
        MMRLambda: 0.7,
        LimitDefault: 30,
        PoolCovisitPerSeed: 50, PoolContentPerSeed:50, PoolTrending:200,
        SeedsFromUserHistory: 5, FreshnessHours: 168, CacheTTL:
10*time.Minute,
    },
}
mux:= http.NewServeMux()
h.Routes(mux)
http.ListenAndServe(":8080", mux)
}

```

Phase 5 – Multi-tenant onboarding flow (2–3 days per domain)

0) Config (experiment registry)

```

# config/experiments.yaml
experiments:
- id: "reco_v1"
  tenant_ids: ["acme-ott","acme-consult","acme-pharma"]
  start_utc: "2025-08-20T00:00:00Z"
  end_utc: "2025-09-30T23:59:59Z"
  variants:
    - key: "A"      # control
      weights: { covisit: 0.0, trending: 0.7, content: 0.3, fresh_bonus: 0.1, seen_penalty: 0.1 }
      mmr_lambda: 0.0 # no MMR
    - key: "B"      # treatment

```

```
weights: { covisit: 0.45, trending: 0.30, content: 0.15, fresh_bonus: 0.1, seen_penalty: 0.1 }
mmr_lambda: 0.7
traffic_split: [50, 50] # percent per variant
unit: "user"           # bucket-by user or account
cache_ttl_seconds: 600
```

1) Assignment (sticky, deterministic, multi-tenant)

```
// exp/assign.go
package exp

import (
    "crypto/sha1"
    "encoding/binary"
)

func bucket(tenant, explID, unitID string, variants []string, split []int) string {
    h := sha1.Sum([]byte(tenant + "|" + explID + "|" + unitID))
    n := binary.BigEndian.Uint32(h[:4]) % 100
    acc := 0
    for i, pct := range split {
        acc += pct
        if int(n) < acc { return variants[i] }
    }
    return variants[len(variants)-1]
}
```

Cache the result for 30d in Redis to keep it extra sticky:

```
key = ab:{tenant}:{explID}:{unitID} -> variant.
```

2) Wire into Recommend API

- On each `/v1/recommend?tenant_id=T&user_id=U...` call:

1. Resolve expID (e.g., `reco_v1`) if within dates & tenant in scope.
2. Get variant via Redis->fallback to `bucket(...)`.
3. Load weights / mmr from the experiment config for that variant.
4. Log an exposure document (below).
5. Include `{exp_id, variant}` in the JSON response so clients can echo it back on click/play events.

Exposure log shape

```
{
  "tenant_id": "acme-ott",
  "exp_id": "reco_v1",
  "variant": "B",
  "user_id": "U123",
  "context": "home",
  "req_ts": "2025-08-19T12:00:01Z",
  "served": [
    {"c_id": "C11", "rank": 1, "score": 1.24, "why": {"covisit": 0.7, "trending": 0.3}},
    {"c_id": "C25", "rank": 2, "score": 1.18, "why": {"covisit": 0.6, "content": 0.4}}
  ],
  "limit": 30,
  "request_id": "r-8f3c..." // generate; return to client
}
```

Store in Mongo collection `reco_exposures` (index: `{tenant_id, exp_id, variant, req_ts}` + `{tenant_id, user_id, req_ts}`).

3) Outcome events (click/play/convert attribution)

Ask clients to send outcomes with `request_id` (or `exp_id + variant + ts` if you can't modify clients yet). Minimal event:

```
{
  "tenant_id": "acme-ott",
  "request_id": "r-8f3c...",
  "user_id": "U123",
  "item_id": "C11",
  "event_type": "click", // "play_60s", "download", "register", "contact_submit"
  "ts": "2025-08-19T12:00:10Z"
}
```

Ingest to `interactions` as today. For robust join, also mirror to a small `reco_outcomes` collection keyed by `request_id` (index `{tenant_id, request_id, ts}`).

Attribution rule: attribute outcomes to the latest exposure for that user/context within X minutes (e.g., 60m), or use `request_id` join if available.

4) Daily KPI job (Mongo pipelines)

Create a cron job `cmd/analytics/daily.go` that runs for each tenant/experiment the previous UTC day.

CTR / Action rate

```
// Pseudocode with aggregation stages
// 1) explode exposures -> rows(user, item, rank, variant)
db.reco_exposures.aggregate([
  {$match: {tenant_id:T, exp_id:"reco_v1", req_ts: {$gte: D0, $lt: D1}}},
  {$unwind: "$served"},
  {$project: {
    tenant_id:1, exp_id:1, variant:1, user_id:1, request_id:1, req_ts:1,
    item_id:"$served.c_id", rank:"$served.rank"
  }},
  // 2) left join outcomes (click, play_60s, domain goal)
  {$lookup:{
    from:"interactions",
    let:{tid:"$tenant_id", uid:"$user_id", iid:"$item_id", t0:"$req_ts"},
    pipeline:[
      {$match: {$expr:{
        $and:[
```

```

    {Seq:["$tenant_id","$$tid"]},
    {Seq:["$u_id","$$uid"]},
    {Seq:["$c_id","$$iid"]},
    {$gte:["$ts","$$t0"]},
    {$lt:["$ts",{ $add:["$$t0", 1000*60*60]}} // within 60m
  ]
  }},
  {$match: {event_type: {$in:["click","play_60s","download","register","contact_submit"]}}},
  {$project:{event_type:1}}
],
as:"evs"
}},
{$addFields:{
  clicked: {$gt:[{$size: {$filter:{input:"$evs",as:"e",cond:{$eq:["$$e.event_type","click"]}}}},0]},
  played60: {$gt:[{$size: {$filter:{input:"$evs",as:"e",cond:{$eq:["$$e.event_type","play_60s"]}}}},0]},
  goal: {$gt:[{$size:
{$filter:{input:"$evs",as:"e",cond:{$in:["$$e.event_type",["download","register","contact_submit"]}}}},
,0]}
}},
{$group:{
  _id:{variant:"$variant"},
  served:{$sum:1},
  clicks:{$sum:{$cond:["$clicked",1,0]}},
  play60:{$sum:{$cond:["$played60",1,0]}},
  goals: {$sum:{$cond:["$goal",1,0]}}
}},
{$project:{
  variant:"$_id.variant",
  served:1,
  ctr: {$divide:["$clicks","$served"]},
  play60_rate: {$divide:["$play60","$served"]},
  goal_rate: {$divide:["$goals","$served"]}
}}
])

```

Watch time / dwell (OTT or content sites)

Join to interactions with `play_seconds` and compute average per served exposure where a play occurred.

Diversity@K & Freshness@K

Compute from the `served` array:

- Diversity: % of unique `series_id/industry` among top K.
- Freshness: % items that appeared in `popular_day` last N days.

Write results to `daily_kpis`:

```
{
  "tenant_id":"acme-ott",
  "date":"2025-08-19",
  "exp_id":"reco_v1",
  "variant":"B",
  "served": 1_240_000,
  "ctr": 0.128,
  "play60_rate": 0.083,
  "goal_rate": 0.012,
  "avg_watch_sec": 214.3,
  "diversity_at_30": 0.78,
  "freshness_at_30": 0.64
}
```

Indexes: `{tenant_id, date, exp_id, variant}`.

5) Client/SDK contract (simple & robust)

- Server response from `/recommend` includes:
 - `request_id, exp_id, variant, items[]`.
- Client sends `request_id` on click/play/goal events (best)
or echoes `exp_id+variant` if you can't add `request_id` yet.

This makes attribution deterministic and simplifies analytics.

6) Rollout & guardrails

- Ramp policy:
 - Day 1: 10% traffic (B), watch errors/latency.
 - Day 2: 25% if stable.
 - Day 3: 50% if KPIs non-negative.
- Auto-kill switch:
 - If 30-min moving CTR drops > X% vs control, or error rate > 1%, set Redis flag `exp:reco_v1:paused=true`; API routes all to control.
- Power check (quick sanity):
 - If baseline CTR $\approx 10\%$ and you expect +5% relative lift, you need $\approx >1-2\text{M}$ served impressions per arm to be confident at 95%/80% power. If you have less, extend the run.

7) Minimal code touches (summary)

- Add assignment (+ Redis cache) in your Recommend handler.
- Expose `request_id`, `exp_id`, `variant` in the response.
- Log exposures (`reco_exposures`).
- Ensure outcome events carry `request_id` (or join window).
- Add daily cron to compute KPIs \rightarrow `daily_kpis`.

- (Optional) Export `daily_kpis` CSV to your BI tool.
-

8) Tiny Go helpers you can paste

```
// exp/store.go (Redis-backed sticky assignment)
func Assign(redis *redis.Client, tenant, expID, user string, variants []string, split []int, ttl
time.Duration) (string, error) {
    key := "ab:" + tenant + ":" + expID + ":" + user
    if v, err := redis.Get(context.Background(), key).Result(); err == nil && v != "" {
        return v, nil
    }
    v := bucket(tenant, expID, user, variants, split)
    _ = redis.Set(context.Background(), key, v, ttl).Err()
    return v, nil
}

// logging.go
func LogExposure(db *mongo.Database, doc any) { _, _ =
db.Collection("reco_exposures").InsertOne(context.Background(), doc) }
```

Phase 6 — Reliability, security, governance (ongoing)

Wraps the engine with health checks, logging, rate limits, circuit breakers, and Redis job locks for safety.

Adds HMAC auth, key rotation, consent gates, audit logs, retention/TTL, metrics, alerts, and backups, making the system production-grade and multi-tenant compliant.

Reliability

1) Health, readiness, liveness

Expose 3 endpoints. Readiness pings Mongo + Redis; Liveness is cheap.

```
// infra/health.go
package infra

import (
    "context"
    "encoding/json"
    "net/http"
    "time"

    "github.com/redis/go-redis/v9"
    "go.mongodb.org/mongo-driver/mongo"
)

type Health struct{ DB *mongo.Database; RDB *redis.Client }

func (h *Health) Routes(mux *http.ServeMux) {
    mux.HandleFunc("/healthz", func(w http.ResponseWriter, _ *http.Request) {
w.WriteHeader(200) })
    mux.HandleFunc("/readyz", h.ready)
    mux.HandleFunc("/livez", func(w http.ResponseWriter, _ *http.Request) {
w.Write([]byte("ok")) })
}

func (h *Health) ready(w http.ResponseWriter, r *http.Request) {
    ctx, cancel := context.WithTimeout(r.Context(), 200*time.Millisecond); defer cancel()
    errM := h.DB.Client().Ping(ctx, nil)
    errR := h.RDB.Ping(ctx).Err()
    resp := map[string]any{"mongo": errM==nil, "redis": errR==nil}
    code := 200; if errM!=nil || errR!=nil { code = 503 }
    w.Header().Set("Content-Type","application/json"); w.WriteHeader(code)
    _ = json.NewEncoder(w).Encode(resp)
}
```

2) Panic recovery + request logging

Structured logs with correlation IDs.

```
// middleware/http.go
package middleware

import (
    "log"
    "math/rand"
    "net/http"
    "time"
)

func RequestID(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        id := r.Header.Get("X-Request-ID")
        if id == "" { id = time.Now().Format("20060102T150405") + "-" + randSeq(6) }
        w.Header().Set("X-Request-ID", id)
        next.ServeHTTP(w, r.WithContext(withReqID(r.Context(), id)))
    })
}

func Recover(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        defer func(){ if rec:=recover(); rec!=nil {
            log.Printf("panic req=%s err=%v", ReqID(r.Context()), rec)
            http.Error(w, "internal", 500)
        }}()
        next.ServeHTTP(w, r)
    })
}

// very small logger
func AccessLog(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        start := time.Now(); next.ServeHTTP(w, r)
        log.Printf("rid=%s method=%s path=%s dur_ms=%d ua=%q",
            ReqID(r.Context()), r.Method, r.URL.Path, time.Since(start).Milliseconds(),
            r.UserAgent())
    })
}
```

3) Rate limiting (per tenant/app/route)

(You already have the idea; here's a complete snippet.)

```
// middleware/ratelimit.go
package middleware

import (
    "fmt"
    "net/http"
    "time"
    "github.com/redis/go-redis/v9"
    "context"
)

func RateLimit(rdb *redis.Client, limitPerMin int) func(http.Handler) http.Handler {
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
            tid, app := r.URL.Query().Get("tenant_id"), r.URL.Query().Get("app_id")
            key := fmt.Sprintf("rl:%s:%s:%s:%d", tid, app, r.URL.Path,
time.Now().Unix()/60)
            n, _ := rdb.Incr(context.Background(), key).Result()
            if n == 1 { _ = rdb.Expire(context.Background(), key, time.Minute).Err() }
            if int(n) > limitPerMin {
                http.Error(w, "rate limit", http.StatusTooManyRequests); return
            }
            next.ServeHTTP(w, r)
        })
    }
}
```

4) Circuit breakers & backoff (Mongo/Redis)

Keep it simple: short timeouts, one retry with jitter, and fallbacks.

```
// infra/retry.go
package infra
```

```
import (
```

```

        "time"
        "math/rand"
    )

    func RetryOnce(delay time.Duration, f func() error) error {
        if err := f(); err != nil {
            time.Sleep(delay + time.Duration(rand.Intn(50))*time.Millisecond)
            return f()
        }
        return nil
    }
}

```

Use in calls that read precomputes; if they fail, fallback to Trending only.

5) Job orchestration safety (single runner)

Prevent duplicate precompute runs using a Redis lock.

```

// jobs/lock.go
package jobs

import (
    "context"
    "time"
    "github.com/redis/go-redis/v9"
)

func WithLock(rdb *redis.Client, key string, ttl time.Duration, fn func() error) error {
    ok, err := rdb.SetNX(context.Background(), key, "1", ttl).Result()
    if err != nil || !ok { return nil } // someone else holds the lock
    defer rdb.Del(context.Background(), key)
    return fn()
}

```

Run like:

```

_ = WithLock(rdb, "lock:covisit:acme-ott:web", 30*time.Minute, func() error {
    return reco.RunCoVisit(ctx, db, "acme-ott", cfg, time.Now())
})

```

6) TTL + retention (governed data lifecycle)

Use Mongo TTL indexes for interactions/idempotency.

```
// util/ttl.go
_, _ = db.Collection("interactions").Indexes().CreateOne(ctx,
    mongo.IndexModel{ Keys: bson.D{{"ts",1}}, Options:
options.Index().SetExpireAfterSeconds(60*60*24*365) }) // 365d
_, _ = db.Collection("idempotency_keys").Indexes().CreateOne(ctx,
    mongo.IndexModel{ Keys: bson.D{{"created_at",1}}, Options:
options.Index().SetExpireAfterSeconds(60*60*48) }) // 48h
```

Security

1) HMAC auth (already added) + key rotation

Support multiple key versions via header `X-Key-Id`.

```
// security/keys.go
type Key struct{ ID string; Secret []byte; Active bool }
func GetTenantKeys(db *mongo.Database, tenant string) []Key { /* fetch from tenants.keys[] */
return nil }
```

Verify by trying the active key first; accept previous for a grace window.

2) IP allowlists per tenant/app (optional)

Add `apps.allowed_cidrs` and reject if source IP not matched.

3) PII minimization & hashing

Store only opaque IDs. If you must store emails/phone for certain tenants:

- Hash before store: `sha256(tenant_id|value)` and drop raw value post-join.
- Mask in logs.

4) Secrets management

- Local: env vars / file with strict perms.
- Azure: Key Vault for Redis key, HMAC secrets. Rotate quarterly; reload on change (poll or webhook).

5) Transport security

- Force TLS for public endpoints (behind your LB).
- Redis (Azure) already TLS on 6380.
- Mongo: enable TLS if remote.

6) RBAC (admin APIs)

- `/admin/*` protected by JWT/OIDC role (`owner`, `ops`, `viewer`).
- Only `owner/ops` can create tenants/apps or rotate keys.

Governance & Compliance

1) Consent & purpose limitation

Attach purpose tags to features and check before use.

```
// policy/consent.go
type Consent struct{ Personalization bool; Ads bool }
func LoadConsent(userID string) Consent { return Consent{Personalization:true} } // stub
```

// in Recommend: if !consent.Personalization => return only generic Trending.

2) Audit log (immutable)

Log what you served and why (per tenant's policy).

```
// audit/log.go
type Audit struct{
  TenantID, AppID, UserID, RequestID string
  Items []struct{ CID string; Score float64; Why map[string]float64 }
  Policies []string
  Ts time.Time
}
func Write(db *mongo.Database, a Audit){ _, _ =
db.Collection("reco_audit").InsertOne(context.Background(), a) }
```

Index: {tenant_id, ts}; TTL as required (e.g., 90d).

3) Data subject requests (delete/export)

- Keep a mapping `user_aliases`: {tenant_id, app_id, user_id, hashed_user_id}.
- Delete path: find by alias, delete interactions for that user, and purge exposures/outcomes/audit (soft-delete if needed).
- Build a small `cmd/gdpr/delete_user.go` using `DeleteMany`.

4) Retention policy per tenant/app

Put in `apps` doc: `retention_days_interactions`, `retention_days_audit`. Drive TTL from there.

Observability & SLOs

1) Metrics (Prometheus)

Minimal counters/histograms.

```
// telemetry/metrics.go
var (
    reqDur = prometheus.NewHistogramVec(prometheus.HistogramOpts{
        Name: "http_request_duration_ms", Buckets:
[]float64{5,10,20,50,100,200,500,1000},
        }, []string{"route"})
    errCtr = prometheus.NewCounterVec(prometheus.CounterOpts{
        Name: "errors_total",
        }, []string{"route","code"})
    cacheHit = prometheus.NewCounterVec(prometheus.CounterOpts{
        Name: "cache_hits_total",
        }, []string{"route"})
)
// expose /metrics via promhttp.Handler()
```

Alert ideas:

- P95 `/v1/recommend` > 60ms for 5m.
- Error rate > 0.5% for 5m.
- Cache hit < 50% for 15m.
- Precompute job duration > SLO.
- Redis/Mongo ping failing.

2) Tracing (OpenTelemetry)

Instrument request path and precompute jobs; sample at 1–5% in prod.

Runbooks & DR

- Backups
 - Mongo: daily snapshots + weekly `mongodump` for logical restore; test restores monthly.
 - Redis: Enterprise persistence or just reconstructable cache (treat as volatile).
- Disaster Recovery
 - Keep infra-as-code; document restore steps (create DB, restore dumps, re-run precompute, warm caches).
- Chaos drills
 - Kill Redis: service should degrade to Trending only.
 - Slow Mongo: circuit breaker → short timeout, retry once, fallback.

Putting it together (server wiring)

```
mux := http.NewServeMux()
health := &infra.Health{DB: db, RDB: redis}
health.Routes(mux)

rec := &api.RecoHandler{DB: db, RDB: redis, Cfg: cfgFromAppOrFile()}
mux.Handle("/v1/recommend",
  middleware.RequestID(
    middleware.Recover(
      middleware.RateLimit(redis, 600)( // 600 rpm example
        middleware.AccessLog(http.HandlerFunc(rec.Recommend)),
      )))

// expose /metrics if you use Prometheus
mux.Handle("/metrics", promhttp.Handler())
```