CHAPTER 1 — MASTER CLASSIFICATION SYSTEM (V5 CANON)

The Global Luxury Retail Ontology powering Tailor Shift
(Fully rewritten, maison-agnostic, refined, and aligned with all optimisations we discussed.)

---

1. Purpose — Why the MCS Exists

The **Master Classification System (MCS)** is the **foundational ontology** of Tailor Shift.

It defines how the luxury retail world is structured, understood, and computed across:

- the data model (Chapter 2),
- the Matching Engine (Chapter 15),
- the Assessment Engine (Chapter 16),
- the Learning Engine (Chapter 17),
- the Personalisation Engine (Chapter 20),
- the Career Projection Engine (Chapter 21),
- all UX journeys (Chapter 3),
- all Brand/Talent interactions (Chapter 19),
- the full technical architecture (Chapter 22–32).

It is **brand-agnostic**, but calibrated on the shared behavioural, operational, product, and service expectations observed in **high-end global luxury retail environments**, including:

- multi-division flagship stores,
- high-touch clienteling environments,
- high-value product categories (RTW, Leather Goods, Watches, High Jewelry, Beauty, Fragrance),

- rigorous operational structures,
- advanced service rituals,
- high-traffic and high-complexity store formats.

The MCS is **not an opinion**.
It is the **structural truth** of luxury retail, organised into a system that machines and humans can reason with.

---

2. Architecture — The 7 Macro-Pillars of the MCS

The MCS is built around **seven intertwined pillars**:

1. **Groups & Houses**
2. **Divisions & Product Architecture**
3. **Role Ladder (Levels 1–8)**
4. **Store Typology & Complexity Tiers**
5. **Experience Taxonomy (FOH / BOH / Leadership / Clienteling / Ops / Business)**
6. **Capability Zones (7 Core Luxury Capabilities)**
7. **Geography & Market Structure**

These pillars interlock:

- Divisions influence the required skills.
- Role level changes behavioural expectations.
- Store complexity reshapes leadership and operational requirements.
- Geography defines pace, norms, and mobility.
- Experience Blocks reveal what a Talent can *actually* do.

This is why Tailor Shift can model the luxury retail world with precision.

---

3. Groups & Houses — Mapping the Luxury Retail Ecosystem

3.1 Global Groups

Used for:

- Talent preference modelling
- House-side filtering
- Matching (Group Fit signals)
- Learning & Projection
- Brand segmentation

Canonical groups:

- LVMH
- Kering
- Richemont
- Capri Holdings
- Tapestry
- Estée Lauder Companies
- L’Oréal Luxe
- Shiseido
- Puig

3.2 Independent and Heritage Houses

Independent luxury houses often share:

- high aesthetic standards,
- more codified brand behaviour,
- specific service rituals,
- distinctive product culture,
- high expectations in image discipline and clienteling.

Impact on engine logic:

- higher Brand Fit weighting,
- higher Clienteling/Service expectations,
- stricter Division Fit in certain categories (e.g., High Jewelry, Watches).

---

4. Divisions — Cross-Brand Product & Skill Architecture

Luxury retail operates across tightly-defined product divisions.

4.1 Primary Divisions

- Ready-to-Wear (RTW)
- Leather Goods
- Shoes
- Watches
- High Jewelry
- Fragrance
- Beauty / Skincare

- Eyewear
- Accessories

4.2 Hybrid Divisions

- Home / Lifestyle
- Digital Accessories

4.3 Skills & Behaviour by Division

Division	Required literacy
RTW	silhouette logic, styling, fitting rituals
Leather Goods	craftsmanship awareness, materials, product architecture
Shoes	fit mastery, comfort expertise
Watches	technical understanding, movement basics
High Jewelry	gemstone literacy, high-precision storytelling
Beauty	sensorial expertise, routine guidance
Fragrance	olfactory narrative, short-form storytelling

Impact: Division Fit is not semantic. It is **behavioural and operational**.

---

5. Role Ladder (Level 1 → Level 8)

A single, house-agnostic spine for all roles worldwide.

Level	Canonical Role	Essence
1	Sales Advisor	FOH basics, service rituals
2	Senior Advisor	anticipation, clienteling foundations

3	Floor/Department Manager	zone coordination, micro-coaching
4	Assistant Store Manager	operational leadership (N-1)
5	Store Manager	full boutique ownership
6	Flagship/XXL Director	multi-division governance
7	Cluster / Area Manager	multi-boutique management
8	Retail / Regional Director	strategic retail leadership

The ladder defines:

- expected behaviours,
- capability maturity per level,
- assessment interpretation,
- career trajectory,
- matching Role Fit.

---

6. Store Typology — The Operational Reality of Luxury Retail

Store types fundamentally change the **behaviours and skills needed**.

Typology (canon):

- Flagship / XXL Store
- Main Boutique
- Department Store Corner
- Travel Retail
- Outlet

- Pop-up
- Partner / Franchise
- High Jewelry Salon
- Beauty Boutique / Beauty Flagship

Each type influences:

- service rhythm,
- operational load,
- client profile,
- staff structure,
- division adjacency,
- leadership requirements.

---

7. Complexity Tiers (T1 → T5)

How Tailor Shift quantifies boutique complexity.

T1 — XXL Flagship

Highest complexity:

- 40–150 FOH staff
- multi-division
- high VIP density
- multi-shift governance
- advanced BOH structures
- peak traffic

- cross-functional coordination (VM, Ops, Security, After-Sales)

T2 — Major Boutique

High demand:

- several divisions
- heavy traffic
- strong FOH/BOH coordination
- leadership layers present

T3 — Standard Boutique

Moderate complexity:

- 1–2 divisions
- concentrated FOH
- generalist skill pattern

T4 — Resort / Seasonal

Volatile context:

- extreme traffic peaks
- high emotional load
- multilingual patterns
- rapid pacing

T5 — Outlet

OPS-intense:

- stock/cash mastery

- high cadence
- lower storytelling / higher productivity focus

---

8. Experience Taxonomy — The Real Work of Luxury Retail

Experience Blocks encode **what a Talent has actually done**, not what is written on a CV.

6 canonical families:

1. **FOH** — service discipline, presence, storytelling, floor control
2. **BOH** — stock accuracy, cash discipline, repairs, organisation
3. **Leadership** — micro-coaching, floor stabilization, prioritisation
4. **Clienteling** — CRM discipline, follow-up timing, segmentation, VIP handling
5. **Operations** — SOPs, cross-functional flow, cadence, execution
6. **Business** — KPIs, productivity, sell-through, planning

They fuel:

- Experience Fit
- Capability Fit
- Assessment
- Learning recommendations
- Projection & readiness

---

9. Capability Zones — The 7 Luxury Retail Capabilities

1. **Service Excellence**
2. **Clienteling & CRM Discipline**
3. **Brand Ambassadorship**
4. **Leadership & People Development**
5. **Operations Management**
6. **Business Performance**
7. **Innovation & Agility (digital fluency)**

Each capability evolves from Level 1 → 8 (detailed in Chapter 10).

Impact:
Capability Fit = **25%** of the matching score.

---

10. Geography Model — Global Luxury Markets

Regions:

- EMEA
- Americas
- APAC
- Middle East

Key hubs:

- Paris, London, Milan

- Dubai, Singapore, Hong Kong
- Shanghai, Tokyo
- New York, Los Angeles

Geography affects:

- mobility,
- Talent preferences,
- store structures,
- division mixes,
- customer patterns.

---

11. The MCS as a Unified Operational Ontology

The MCS is:

- a **graph** of the luxury retail world,
- a **behavioural ontology** (not keyword-based),
- a **predictive engine input**,
- a **classification system** aligned with real boutique environments,
- the foundation for a **future luxury-specific LLM**.

It ensures that every Talent and every Opportunity is described using **the same global language**.

Without the MCS, Tailor Shift cannot reason, match, assess, or project.

CHAPTER 2 — ENTITIES & DATA STRUCTURES

The canonical database & domain model for Tailor Shift
(Fully rewritten, clean, neutral, and aligned with the global luxury retail ontology.)

---

0. Purpose — Why Entities Matter

The domain entities defined here:

- encode **all information Tailor Shift understands**,
- power the **matching, assessment, learning, projection, and personalisation engines**,
- drive every **Talent** and **Brand** journey,
- structure all **API routes**,
- define the **Tiger Cloud PostgreSQL schema**,
- and form the **semantic backbone** for future AI reasoning.

They are:

- **brand-agnostic**,
- **global**,
- **fully normalized**,
- **behaviourally rich**,
- and **compatible with the luxury retail ontology** (Classifications, Role Ladder, Store Tiers, Experience Blocks, Capability Model).

Every UI screen (Talent or Brand) maps directly onto these entities.

---

1. Overview — The 12 Canonical Entities

Tailor Shift models luxury retail using **12 core entities**, each precise and indispensable:

1. **UserAccount**
2. **Talent**
3. **Brand / House**
4. **Store**
5. **Opportunity**
6. **NormalizedRole (Role Ladder)**
7. **ExperienceBlock**
8. **Assessment**
9. **LearningModule**
10. **Geography**
11. **Category / Division**
12. **StoreComplexityIndex**

These entities are:

- **coherent with the MCS (Chapter 1)**,
- **explicitly typed**,
- **designed for fast queries**,
- **future-compatible with LLM-based engines**,
- and consistent across Talent, Brand, Matching and UI journeys.

---

2. ENTITY DEFINITIONS — Tailor Shift Canon (V5)

Below is the **entire rewritten entity specification**, integrating:

- compensation modelling (confidential talent data + house-side package bands),
- privacy-by-design rules,
- behavioural depth via Experience Blocks,
- role mapping,
- all matching dimensions,
- assessment & learning integration,
- clean Tiger Cloud PostgreSQL structure.

2.1 USERACCOUNT — System Identity

Represents a **login identity** in Auth.js.

UserAccount {

- `id: UUID`
- `email: string`
- `password_hash: string`
- `role: 'talent' | 'brand_admin' | 'recruiter' | 'admin'`
- `linked_talent_id: UUID | null`
- `linked_brand_id: UUID | null`
- `created_at, updated_at`
  }

Purpose:

- single identity per person,
- drives RBAC & route protection,
- connects user → Talent or Brand entity.

2.2 TALENT — The Central Entity

The most detailed and behaviourally rich entity in the system.

It encodes:

- identity,
- experience,
- preferences,
- assessment,
- learning progress,
- compensation (confidential),
- matching signals,
- behavioural richness via Experience Blocks.

Talent {

Identity & Contact

- `id: UUID`
- `account_id: UUID`
- `first_name, last_name: string`

- `email: string` *(Talent's email is never exposed to Brands until Talent approves an introduction)*

Location

- `location: Geography`
- `mobility: { open_to_relocation: boolean, preferred_regions: region_enum[], preferred_cities: string[] }`

Professional Profile

- `headline: string`
- `bio: string | null`
- `years_experience: number`
- `current_role_level: number` *(1–8 from the Role Ladder)*
- `official_title: string`
- `current_brand_id: UUID | null`
- `current_store_id: UUID | null`
- `divisions_experience: division_enum[]`

Openness Status

- `openness_status: openness_enum (actively_looking | open_to_opportunities | not_looking)`
- `openness_comment: string | null`

Career Preferences
(used for Matching Preference Fit)

- `career_preferences: {`

- `target_groups: string[]`
- `target_brands: string[]`
- `target_levels: number[]`
- `target_divisions: division_enum[]`
- `target_store_types: store_type_enum[]`
- `target_regions: region_enum[]`
- `target_countries: string[]`
- `target_cities: string[]`
  `}`

Languages

- `languages: [{ code: string, level: 'basic'|'intermediate'|'fluent'|'native' }]`

Experience

- `experience_blocks: ExperienceBlock[]` *(FOH/BOH/Leadership/Ops/Business/Clienteling blocks)*

Assessment

- `assessment_summary: { service_excellence, clienteling, brand_ambassadorship, leadership_signals } | null`
- `assessment_ids: UUID[]`

Learning

- `completed_learning_modules: UUID[]`
- `badges: JSON[]`

Confidential Compensation (NEW)

- `compensation_profile: {`
    - `current: { base, variable, allowance }`
    - `visibility: 'internal_only'`
    - `last_updated: datetime`
      `}`

Interactions

- `interested_opportunities: UUID[]`

Timestamps

- `created_at, updated_at`

}

Notes:

- Compensation is **never exposed** to Brands.
- Only **derived alignment signals** (within ±25%) are used in matching.

2.3 BRAND — Employer Identity

Represents a maison or luxury house.

Brand {

- `id: UUID`
- `name: string`
- `group_type: string` *(Group / Independent etc.)*

- `is_independent: boolean`
- `divisions: division_enum[]`
- `hq: Geography`
- `markets: region_enum[]`
- `employer_page: { headline, description, mission, website, video_url }`
- `stores: Store[]`
- `created_at, updated_at`
  }

Notes:

- Brand-side admin/recruiter accounts link here.
- Employer page fields power the **Brand Profile**.

2.4 STORE — Operational Unit

The boutique is the **core place where luxury retail happens**.

Store {

- `id: UUID`
- `brand_id: UUID`
- `name: string`
- `geography: Geography`
- `store_type: store_type_enum`

- `size_class: store_size_enum`
- `complexity_tier: complexity_tier_enum` *(T1–T5)*
- `divisions_present: division_enum[]`
- `ownership: 'direct' | 'franchise' | 'partner'`
- `footfall_profile: 'high' | 'medium' | 'low' | null`
- `notes: string | null`
- `created_at, updated_at`
  }

2.5 NORMALIZED ROLE — Canonical Ladder Mapping

Maps any job title to a ladder level 1–8.

NormalizedRole {

- `id: int`
- `level: int`
- `label: string`
- `synonyms: string[]`
  }

Defines how an “Assistant Boutique Manager” maps into level 4, etc.

2.6 EXPERIENCEBLOCK — Behavioural Work Units

Captures granular, observable experience.

ExperienceBlock {

- `id: UUID`
- `talent_id: UUID`
- `block_type: 'FOH' | 'BOH' | 'Leadership' | 'Clienteling' | 'Operations' | 'Business'`
- `description: string`
- `achievements: string[]`
- `years_in_block: number`
- `created_at: datetime`
  }

Used directly in:

- Experience Fit
- Capability Fit
- Assessment interpretation
- Projection

2.7 OPPORTUNITY — Contextualised Luxury Role

Opportunity {

Identity

- `id: UUID`
- `brand_id: UUID`
- `store_id: UUID | null`
- `title: string`
- `normalized_role_level: int`
- `description: string`

Requirements (JSONB)

- `min_years_experience: number | null`
- `required_levels: number[]`
- `divisions_required: division_enum[]`
- `languages_required: string[]`
- `store_type_experience: store_type_enum[]`
- `category_experience: division_enum[]`
- `geography: Geography | null`

Assessment thresholds

- `min_assessment_scores: { service_excellence?, clienteling?, brand_ambassadorship?, leadership_signals? }`

Compensation band (NEW)

- `compensation: { base_min, base_max, variable_target, allowance }`

Matching

- `match_results: [{ talent_id, match_score }]` *(cache)*

Status

- `status: 'open' | 'closed' | 'paused'`

Timestamps

- `created_at, updated_at`
  }

2.8 ASSESSMENT — Retail Excellence Scan Result

Assessment {

- `id: UUID`
- `talent_id: UUID`
- `version: string`
- `answers: JSONB` *(per question)*
- `scores: { service_excellence, clienteling, brand_ambassadorship, leadership_signals }`
- `insights: JSONB` *(short behavioural summaries)*
- `created_at, updated_at`
  }

Used in:

- Matching (Capability Fit),

- Talent Dashboard,
- Projection,
- Learning Engine.

2.9 LEARNINGMODULE — Structured Micro-Learning Unit

LearningModule {

- `id: UUID`
- `title: string`
- `category: 'service' | 'clienteling' | 'leadership' | 'operations' | 'brand' | 'business' | 'innovation'`
- `duration_minutes: number`
- `summary: string`
- `content_type: 'slides' | 'text' | 'video'`
- `content_ref: string`
- `quiz: JSONB`
- `created_at, updated_at`
  }

2.10 GEOGRAPHY — Normalized Location Descriptor

Geography {

- `region: region_enum`
- `country: string`
- `city: string`
  }

Used consistently in Talent, Store, Opportunity.

2.11 CATEGORY / DIVISION — Product Families

Simple structured definition for all divisions.

Category {

- `id: UUID`
- `label: division_enum`
- `description: string`
  }

2.12 STORE COMPLEXITY INDEX — High-Resolution Context

StoreComplexityIndex {

- `store_id: UUID`
- `tier: complexity_tier_enum`

- `traffic: 'high' | 'medium' | 'low'`
- `staff_size: number | null`
- `categories: division_enum[]`
- `notes: string | null`
  }

Feeding:

- Matching (Tier Bonus),
- Projection,
- Learning recommendations.

---

3. RELATIONSHIP MODEL (V5)

UserAccount → Talent

One-to-zero-or-one (depending on role)

UserAccount → BrandAdmin/Recruiter

One-to-zero-or-one

Brand → Stores

One-to-many

Brand → Opportunities

One-to-many

Talent → ExperienceBlocks

One-to-many

Talent → Assessments

One-to-many

Opportunity ↔ Talent

Many-to-many through matching computations

Talent → LearningModules

Many-to-many via `talent_learning_progress`

---

4. AI & Engine Considerations

These entities are structured for high-fidelity reasoning:

- deterministic matching,
- behavioural inference,
- capability weighting,
- FOH/BOH/Leadership/Business signal extraction,
- division-aware behavioural scoring,
- tier-aware operational scoring,
- and future LLM verticalisation.

They are **explainable**, **auditable**, and support:

- low-risk AI classification (EU AI Act),
- transparent matching outputs,
- privacy guarantees,
- bias minimisation.

CHAPTER 3 — USER JOURNEYS (V5 CANON)

The complete end-to-end flow for Talent, Brand, and System journeys
(Fully rewritten, neutral, global, precise, luxury-aligned.)

---

0. Purpose — Why User Journeys Matter

User Journeys describe:

- how a Talent enters the system,
- how a House (Brand) publishes and evaluates opportunities,
- how matching, assessment, learning, and projection connect,
- how privacy, discretion, and pacing are preserved,
- how the interface remains consistent across web and mobile.

They are:

- **strictly tied to the Tiger Cloud data model**,
- **strictly aligned with the luxury UX/UI canon**,
- **strictly dependent on behavioural and operational expectations**,
- **designed to be frictionless and elegant**,
- **the single source of truth** for orchestrating screens, API calls, and engines.

Nothing in Tailor Shift can override or diverge from these journeys.

STRUCTURE

1. Talent Journeys
2. Brand Journeys
3. Matching Journey

4. Account Management
5. Cross-Cutting Technical Constraints

---

1. TALENT JOURNEYS (V5)

A refined global experience for luxury retail professionals.

Talent Journeys govern **exactly** how a professional:

- signs up,
- creates their profile,
- completes their experience,
- takes the Retail Excellence Scan,
- receives opportunities,
- progresses with learning,
- interacts discreetly with houses through controlled introductions.

Every step feeds directly into the data model and the engines.

1.1 Talent Onboarding (Steps 0 → 7)

Sequential, structured, elegant. No skipped steps except Step 5.

Step 0 — Entry Point

URL: `/signup`
Talent selects:

- **“I am a Retail Professional”**

Backend: pre-creates a minimal `UserAccount` structure.
Frontend: routes to Step 1.

Step 1 — Create Account

Inputs:

- email
- password

Backend:

- creates `UserAccount { role='talent' }`
- logs Talent in immediately
- creates a minimal `Talent` record

Redirect: **Step 2**.

Step 2 — Basic Information

Inputs:

- first name / last name
- country / city / region
- languages + proficiency

DB fields:

- `talents.first_name`
- `talents.location`
- `talents.languages`

Impact:

- sets the default **Geography Fit** dimension in matching

- informs future learning modules and mobility logic

Step 3 — Professional Identity

Inputs:

- official job title (free text)
- normalized role (ladder level 1–8)
- years of experience
- divisions experience
- brand/store affiliation (optional)
- store type exposure

Impact:

- anchors the Talent in the **Role Ladder**
- initialises **Role Fit**, **Division Fit**, and **Store Type Fit**
- sets default **preference patterns**

Step 4 — Mobility & Openness

Inputs:

- open_to_relocation
- preferred regions
- openness status (actively looking / open / not looking)
- optional comment

Impact:

- Matching: Geography Fit + Openness Fit
- Talent Dashboard contextualisation
- Brand-side filtering

Step 5 — Career Preferences (Optional but recommended)

Inputs:

- target groups
- target brands
- target role levels
- target divisions
- target store types
- target regions / countries / cities

Impact:

- Matching Preference Fit (10%)
- Filters opportunity suggestions
- Tightly scopes Talent’s visible universe

Step 6 — Assessment Prompt

Talent is invited to start:

- **Retail Excellence Scan** (10–12 behavioural questions)

They may start now or skip to the Dashboard.
Journey remains elegant, not intrusive.

Step 7 — Talent Dashboard

Talent is considered “onboarded”.

Dashboard displays:

- Profile completion bar
- Assessment summary (if completed)
- Recommended learning modules
- Recommended opportunities
- Preferences & openness
- Next steps

The core engines begin working immediately.

1.2 Talent Profile Completion Flow

Activated via:

- onboarding incomplete
- Talent clicks “Edit Profile”

Step 1 — Personal Info

Add:

- phone
- headline
- bio

Step 2 — Experience Blocks

Critical for Matching and Projection.

Inputs per block:

- block type (FOH / BOH / Leadership / Clienteling / Ops / Business)
- description
- achievements
- years in block

Step 3 — Division Mastery

Talent rates their experience per division:

- beginner / intermediate / advanced / expert

Impact on Division Fit + Learning.

Step 4 — Career Preferences Refinement

Finer tuning of preferences.

Step 5 — Publish Profile

Flags profile as complete.

1.3 Talent Assessment Flow (Retail Excellence Scan)

Step 1 — Start Screen

Explains:

- duration
- dimensions measured
- behavioural nature of the assessment

Step 2 — Question Wizard

- 10–12 questions
- one question per screen

- four options (A/B/C/D)
- mobile-first, elegant

Step 3 — Scoring

Backend:

- runs Assessment Engine
- saves `Assessment`
- updates `assessment_summary` in `Talent`

Step 4 — Insights

Per dimension:

- concise behavioural interpretations
- high / medium / low signals

Step 5 — Add to Profile

Assessment becomes visible:

- in Dashboard
- in Talent Profile
- in Brand-side Talent Card (summary only)

Step 6 — Next Steps

- recommended modules
- recommended opportunities
- preference adjustment

1.4 Talent Opportunity Browsing

Step 1 — Entry

`/talent/opportunities`

Step 2 — Listing

Cards show:

- house
- role level
- region
- divisions
- match score
- short requirements

Sorted by match_score desc.

Step 3 — Detail

`/talent/opportunities/{id}`

Shows:

- description (clean, concise)
- requirements
- match breakdown
- division overlap
- store context
- compensation band (house-side only)

Step 4 — Express Interest

CTA: **Send Profile**

Backend:

- Talent added to `interested_opportunities`
- Interaction record created
- Brand sees Talent card but **without contact details**

Privacy preserved.

1.5 Talent Settings

`/talent/settings`

Fields:

- update email
- update password
- edit openness
- edit languages
- delete account (privacy-preserving, full cascade)

---

2. BRAND JOURNEYS (V5)

Designed for speed, structure, and quiet precision.

Brand journeys are built for:

- retail directors,
- area managers,
- HR partners,
- boutique managers,

- regional recruiters.

Experience must be:

- clear,
- structured,
- fast,
- aligned with luxury UX.

2.1 Brand Onboarding Flow

Step 0 — Entry

“I represent a House / Maison.”

Step 1 — Create Account

Email + password
role = `brand_admin`

Backend:

- creates Brand record
- links Account to Brand

Step 2 — Brand Definition

Inputs:

- brand name
- group type
- independent flag
- divisions
- HQ geography

- markets

Step 3 — Employer Page (Optional)

Fields:

- headline
- description
- website
- optional video

Step 4 — Store Declaration (Optional MVP)

Add stores:

- city, country, region
- type
- size
- complexity tier
- divisions
- ownership

Step 5 — Brand Dashboard

Shows:

- active opportunities
- suggested matches
- recent applicants
- CTA: **Post Opportunity**

2.2 Opportunity Posting Flow

Step 1 — Create

`/brand/opportunities/new`

Step 2 — Define Role

- title
- normalized role level
- store (optional)
- description

Step 3 — Requirements

- min years experience
- required levels
- divisions
- languages
- store type experience
- geography
- min assessment scores
- compensation band (NEW)

Step 4 — Preview

Step 5 — Publish

Triggers optional match precompute.

2.3 Brand Talent Search & Match Flow

Step 1 — Search

Filters:

- role level
- divisions
- store type experience
- geography
- assessment thresholds
- openness status
- preference overlap

Step 2 — Talent Cards

Show:

- name
- role level
- divisions
- assessment summary
- preferences
- match score (if tied to an opportunity)

Step 3 — Talent Profile View

Email/phone always masked.

Step 4 — Request Introduction

Creates an interaction pending Talent approval.

Privacy preserved.

---

3. MATCHING JOURNEY

The dual perspective: Talent ↔ Brand.

Step 1 — Opportunity opens

Matching list produced or recomputed.

Step 2 — Suggested matches

Brand sees ranked Talent list.

Step 3 — Brand selects

Shows match breakdown:

- role
- divisions
- store
- geography
- capabilities
- preferences

Step 4 — Request introduction

Talent receives a clear, discreet notification.

Step 5 — Talent side

Talent accepts or declines.
Contact details remain private until acceptance.

---

4. ACCOUNT MANAGEMENT & LOGOUT

Talent

- update personal information
- update preferences
- update openness
- delete account

Brand

- update employer page
- add/remove stores
- manage opportunities
- manage recruiters
- delete brand account

Logout

- clear session
- redirect `/`

---

5. CROSS-CUTTING TECHNICAL CONSTRAINTS

These journeys depend on strict rules:

RBAC

- `/talent/*` → role = talent
- `/brand/*` → role ∈ {brand_admin, recruiter}

DB Integrity

Every field links cleanly to Chapter 2 entities.

Routing

Next.js App Router follows the exact sitemap in Chapter 4.

UX/UI

All screens use the luxury-grade component system (Chapter 5).

Assessment / Matching / Learning

Never computed client-side.

CHAPTER 4 — SITEMAP & ROUTING (V5 CANON)

The complete Next.js App Router structure for Tailor Shift
(Clean, deterministic, global, scalable.)

---

0. Purpose — Why a Canonical Sitemap?

The Sitemap is:

- the **final blueprint** for the Next.js project structure,
- the **map** for all Talent and Brand journeys,
- a **security boundary** (RBAC),
- a **UX architecture plan**,
- the **contract** between Design, Product, and Engineering,
- the **QA checklist**,
- the **foundation** for Antigravity code generation and LLM tooling.

Tailor Shift is a **domain-first** platform.
The Sitemap ensures the code mirrors the domain precisely.

---

1. ROUTING PRINCIPLES (Next.js App Router)

Tailor Shift uses **pure App Router**.

1.1 Folders = Routes

Each folder under `/app` defines a route.

Example:

`/talent/dashboard → /app/(talent)/dashboard/page.tsx`

1.2 Segment-Based Layouts

- `/app/layout.tsx` → global shell
- `/app/(public)/layout.tsx` → public pages
- `/app/(talent)/layout.tsx` → talent pages
- `/app/(brand)/layout.tsx` → brand pages

1.3 RBAC via middleware

- Talent pages require `role = talent`
- Brand pages require `role ∈ {brand_admin, recruiter}`
- Public pages require no session

All enforced in `/middleware.ts`.

1.4 Server Components First

All pages (`page.tsx`) are **Server Components** unless interactivity is required.

1.5 Client Components only when needed

Forms, inputs, assessment wizard, filters.

1.6 API Routes inside `/app/api/*`

Following Next.js App Router pattern:

```
/app/api/opportunities/route.ts
/app/api/assessments/route.ts
/app/api/interactions/[id]/accept/route.ts
```

1.7 SEO

- `/app/robots.txt`
- `/app/sitemap.xml`

Public pages use strong SEO for authority and competence.

---

2. GLOBAL SITEMAP (V5)

Below is the **complete, canonical tree**, with annotations.

2.1 PUBLIC AREA — (public)

Accessible without login.
Minimal, premium, focused.

```
/app
└── (public)
    ├── layout.tsx
    ├── page.tsx                 → Home
    ├── professionals/
    │   └── page.tsx             → "For Professionals"
    ├── brands/
    │   └── page.tsx             → "For Houses & Maisons"
    ├── about/
    │   └── page.tsx
    ├── resources/
    │   ├── page.tsx              → resource hub
    │   └── luxury-retail-competency-framework/
    │       └── page.tsx          → public educational page
    ├── login/
    │   └── page.tsx
    ├── signup/
    │   └── page.tsx
    ├── terms/
    │   └── page.tsx
    ├── privacy/
    │   └── page.tsx
    ├── not-found.tsx
    └── error.tsx
```

Notes:

- Header: clean navigation (Home / Professionals / Houses / Login / Signup).
- Footer: terms, privacy, minimal links.
- No Talent or Brand data appears here.
- `/resources` hosts public educational content and boosts SEO authority.

2.2 TALENT AREA — (talent)

Protected: `role = talent`.

```
/app
└── (talent)
    ├── layout.tsx                → Talent shell (premium, sidebar
when desktop)
    ├── dashboard/
    │   └── page.tsx
    ├── profile/
    │   ├── page.tsx             → overview
    │   ├── edit/page.tsx
    │   ├── experience/page.tsx
    │   ├── preferences/page.tsx
    │   └── skills/page.tsx
    ├── assessment/
    │   ├── page.tsx               → start screen
    │   ├── questions/page.tsx     → client component wizard
    │   ├── results/page.tsx
    │   └── insights/page.tsx
    ├── learning/
    │   ├── page.tsx
    │   └── [moduleId]/page.tsx
    ├── opportunities/
    │   ├── page.tsx
    │   └── [id]/page.tsx
    ├── community/
    │   └── page.tsx               → V2 placeholder
    └── settings/
        └── page.tsx
```

Notes:

- Dashboard = a constellation of engines (matching, assessment, learning, projection).
- Assessment uses a multi-step wizard.
- Opportunity detail includes full match breakdown.
- Contact details remain fully private until Talent accepts a Brand introduction.

2.3 BRAND AREA — (brand)

Protected: `role ∈ {brand_admin, recruiter}`.

```
/app
└── (brand)
    ├── layout.tsx
    ├── dashboard/
    │   └── page.tsx
    ├── profile/
    │   ├── page.tsx
    │   └── edit/page.tsx
```

```
├── stores/
│   ├── page.tsx
│   ├── new/page.tsx
│   └── [id]/page.tsx
├── opportunities/
│   ├── page.tsx
│   ├── new/page.tsx
│   ├── [id]/page.tsx
│   └── [id]/matches/page.tsx
├── search/
│   └── page.tsx              → Talent search with filters
├── talent/
│   └── [id]/page.tsx         → Talent detail (contact masked)
└── settings/
    └── page.tsx
```

Notes:

- Brand Dashboard shows open opportunities, suggested matches, recent applicants.
- Store CRUD is simple but structured.
- Opportunity posting uses a multi-step flow (role → requirements → compensation band → preview).
- Matches page shows ranked talents with match breakdown.
- Talent detail hides contact unless introduction accepted.

2.4 SYSTEM / SHARED PAGES

```
/app
├── onboarding/page.tsx       → dynamic onboarding driver (future
use)
├── terms/page.tsx
├── privacy/page.tsx
├── not-found.tsx
└── error.tsx
```

These pages must follow luxury-grade copy, spacing, and neutral tone.

3. ROUTING → DATA MODEL MAPPING

Every route maps *directly* to canonical entities (Chapter 2) and engines.

Talent Area

Route	Entity	Engine
`/talent/dashboard`	Talent, Assessment, Opportunity	Matching, Learning, Projection
`/talent/profile/*`	Talent, ExperienceBlock	—
`/talent/assessment/*`	Assessment	Assessment Engine
`/talent/learning/*`	LearningModule	Learning Engine
`/talent/opportunities/*`	Opportunity	Matching Engine

Brand Area

Route	Entity	Engine
`/brand/dashboard`	Brand, Opportunity, Talent	Matching
`/brand/stores/*`	Store	—
`/brand/opportunities/*`	Opportunity	Matching
`/brand/search`	Talent	Matching (filtered)
`/brand/talent/[id]`	Talent (public view)	—

4. ROUTING → SECURITY (RBAC)

Enforced in `/middleware.ts` using Auth.js session:

Public

Accessible to all:

- Home / Professionals / Houses / About
- Legal pages
- Login / Signup
- Resources

Talent segment

Allowed roles:
`role = talent`

Else → redirect `/login`.

Brand segment

Allowed roles:
`role ∈ {brand_admin, recruiter}`

Else → redirect `/login`.

Admin (future)

Reserved for internal tools and governance dashboards.

---

5. ROUTING → FRONT-END IMPLEMENTATION

5.1 Server Components default

All `page.tsx` files are server-rendered for:

- faster TTFB,
- secure data fetching,
- predictable UX.

5.2 Client Components only for:

- forms
- multi-step wizards
- filters
- modals

5.3 Data Fetching

No client-side fetching for privileged data.

Use:

```
import { getServerSession } from "next-auth";
import { prisma } from "@/lib/db";
```

5.4 Layout structure

- Global layout: minimal header + neutral spacing
- Talent layout: sidebar on desktop
- Brand layout: navigation blocks

5.5 Revalidation & Caching

Static content revalidates automatically.
Dynamic content is server-rendered every request.

---

6. ROUTING → DESIGN SYSTEM (Chapter 5)

Each page must use the Tailor Shift component system:

- `Card`

- `Input` / `Select` / `Textarea`
- `Tabs`
- `ScoreBar`
- `MatchScore`
- `ProfileCompletionBar`
- `Badge`
- `TalentCard`
- `OpportunityCard`
- `StoreCard`
- `AssessmentCard`

No freestyle UI.
No third-party layout hacks.

---

7. COMPLETE ASCII TREE (Final Canon)

```
/app
├── layout.tsx
├── (public)
│   ├── layout.tsx
│   ├── page.tsx
│   ├── professionals/page.tsx
│   ├── brands/page.tsx
│   ├── about/page.tsx
│   ├── resources/page.tsx
│   ├── resources/luxury-retail-competency-framework/page.tsx
│   ├── login/page.tsx
│   ├── signup/page.tsx
│   ├── terms/page.tsx
│   ├── privacy/page.tsx
│   ├── not-found.tsx
│   └── error.tsx
│
├── (talent)
│   ├── layout.tsx
│   ├── dashboard/page.tsx
│   ├── profile/page.tsx
│   ├── profile/edit/page.tsx
│   ├── profile/experience/page.tsx
│   ├── profile/preferences/page.tsx
│   ├── profile/skills/page.tsx
│   ├── assessment/page.tsx
│   ├── assessment/questions/page.tsx
│   ├── assessment/results/page.tsx
│   ├── assessment/insights/page.tsx
│   ├── learning/page.tsx
│   ├── learning/[moduleId]/page.tsx
│   ├── opportunities/page.tsx
│   ├── opportunities/[id]/page.tsx
│   ├── community/page.tsx
│   └── settings/page.tsx
│
├── (brand)
│   ├── layout.tsx
│   ├── dashboard/page.tsx
│   ├── profile/page.tsx
│   ├── profile/edit/page.tsx
│   ├── stores/page.tsx
│   ├── stores/new/page.tsx
│   ├── stores/[id]/page.tsx
│   ├── opportunities/page.tsx
│   ├── opportunities/new/page.tsx
│   ├── opportunities/[id]/page.tsx
│   ├── opportunities/[id]/matches/page.tsx
│   ├── search/page.tsx
│   ├── talent/[id]/page.tsx
│   └── settings/page.tsx
│
├── onboarding/page.tsx
└── error.tsx
```

CHAPTER 5 — UX / UI GUIDELINES (V5 CANON)

The luxury-grade interface language of Tailor Shift
(Clean, calm, technical, elegant.)

---

0. PURPOSE — WHY A UX/UI CANON?

Luxury retail carries expectations:

- professionalism,
- restraint,
- elegance,
- clarity,
- quality of gesture,
- premium service feeling.

UX/UI must reflect this.

This chapter defines:

- layout systems,
- typography,
- spacing,
- colors,
- visual hierarchy,
- component standards,
- interaction patterns,
- motion rules,

- tone & microcopy,
- mobile-first behaviour,
- accessibility guidelines.

Nothing in implementation may diverge from this system.

---

1. THE UX PHILOSOPHY — “QUIET LUXURY. HIGH SIGNAL. ZERO NOISE.”

Tailor Shift is **not a job board**.
It is a **luxury-grade talent intelligence product**.

Four UX pillars define the experience:

1. Quiet Luxury

- calm, understated elegance,
- refined typography,
- off-white foundations,
- matte accents,
- no gradients,
- no visual gimmicks.

2. High Signal

- every pixel carries meaning,
- every screen has a purpose,
- data and structure come before decoration.

3. Zero Noise

- no clutter,
- no marketing fluff,
- no confusing interface patterns.

4. Discretion & Respect

- privacy-first displays,
- information revealed only when appropriate,
- contact details masked until a Talent explicitly accepts.

This is the **identity layer** of Tailor Shift.

---

2. DESIGN TOKENS — CORE FOUNDATIONS

Tailor Shift uses a very strict, minimal design system.

2.1 Color Palette

Primary

- **Off-White**: `#FAFAF8` (background)
- **Charcoal**: `#1A1A1A` (foreground)
- **Concrete Grey**: `#E0E0DA` (borders/dividers)
- **Pure White**: `#FFFFFF` (cards, panels)

Accent

- **Matte Gold**: `#C2A878` (sparingly used for highlight, not decoration)

Functional Colors

- Success: `#097969`
- Warning: `#C18400`
- Error: `#B40000`
- Info: `#003CBF`

Rules:

- Gold used only to highlight **capability scores**, **role levels**, **match cards**, and subtle decorative marks.
- No bright or neon colors.
- No heavy gradients.
- No drop shadows except ultra-soft overlays.

2.2 Typography

Two-font system:

1. Manrope (UI / Body)

- weights: 300, 400, 500, 600
- used for labels, paragraphs, buttons, lists, numeric displays
- modern, neutral, readable

2. Playfair Display (Editorial)

Used only for:

- large headings
- hero sections
- Talent/Brand profile names

- key narrative moments
- static marketing content

Rules:

- No serif text under 24px.
- No italics except quotes.
- No mixed fonts inside the same paragraph.

2.3 Sizing & Spacing Scale

All spacing must use a strict 4/8-based scale:

`4 / 8 / 12 / 16 / 24 / 32 / 40 / 48 / 64`

Rules:

- No arbitrary pixel spacing.
- Vertical rhythm must follow either:
    - 16 → 24 → 32
    - or 12 → 20 → 28 (mobile contexts)

Large screens → increased padding (16–24px margin sides).
Cards → 24px inner padding.

2.4 Radius, Borders, Shadows

- Border radius: `8px` on interactive UI, `12px` on cards.
- Borders: `1px solid #E0E0DA`.
- Shadows: only ultra-soft overlays such as modals (`rgba(0,0,0,0.08)`).

No drop shadows on cards or buttons.

---

3. COMPONENT SYSTEM — PRIMITIVES

Every UI element is standardised.
No freestyle UI beyond these components.

3.1 Core Components

Card

- white background (#FFFFFF)
- 12px radius
- 1px border
- 24px padding

Section Heading

- Playfair Display
- 24–32px
- bottom margin: 16px

Label

- Manrope
- 12px, medium weight
- uppercase optional (rare, used for metadata)

Input / Select / Textarea

- border: 1px solid concrete grey

- radius: 8px
- padding: 12px
- font-size: 14px
- focus: charcoal border

Button

Variants:

- Primary (charcoal background, white text)
- Secondary (white background, charcoal border)
- Ghost (transparent)

Radius: 8px
Padding: 10px 16px
Font size: 14px
Weight: 500

Tabs

- underline animation on select
- spacing: 16px horizontal

Badge

- small rounded label
- light gold or charcoal border
- used sparingly

ScoreBar

- simple horizontal bar
- background grey

- filled portion in charcoal or gold
- no gradients

MatchScore

- circular numerical display
- 1px charcoal border
- gold percentage text when >70%

---

4. LAYOUT SYSTEM — STRUCTURE FOR EVERY PAGE

Tailor Shift uses a **three-tier layout**:

4.1 Global Layout

- header (public navigation)
- body container (max-width 1200px)
- footer (legal links)

4.2 Talent Layout

- header (simplified)
- optional desktop sidebar (Dashboard, Profile, Opportunities, Learning, Settings)
- mobile-first: top menu tray
- main content area

4.3 Brand Layout

- header with house name
- grid-based dashboard
- emphasis on clarity → tables, cards, structured grids

---

5. PAGE COMPOSITION PATTERNS

Every page must follow one of these canonical patterns:

Pattern A — “Hero + Sections” (Public pages)

Used for:

- Home
- Professionals
- For Houses
- About

Structure:

1. Hero (large serif heading + subtext)
2. Alternating content sections
3. Clear CTA
4. Footer

Pattern B — “Card Grid” (Dashboards)

Used for:

- Talent Dashboard
- Brand Dashboard

Structure:

- 2–3 columns on desktop
- stacked cards on mobile
- each card is atomic and focused

Pattern C — “Form Wizard” (Onboarding, Assessment)

- multi-step
- back / next
- progress indicator
- full-width mobile cards
- light gold progress bar

Pattern D — “Detail Page” (Opportunity, Talent, Brand)

- header section (role/title + metadata)
- content sections (requirements, divisions, store context)
- CTA pinned low on mobile
- match breakdown table

---

6. INTERACTION DESIGN — BEHAVIOUR RULES

6.1 Motion

- minimal
- fade transitions <150ms
- tab underline slide <120ms
- list reordering 0ms (no animation)

6.2 Hover

- slight darkening or border accent
- no color inversion

6.3 Focus

- 2px charcoal outline
- visible keyboard navigation

6.4 Scroll

- soft top shadows on containers
- never hide scrollbars

---

7. MOBILE-FIRST PRINCIPLES

- All screens must be designed **for mobile first**, then expanded.

- Cards stack vertically.
- Forms use simplified spacing.
- Fixed footers allowed for CTAs.
- Navigation collapses into bottom bar or top-right menu.

---

8. PRIVACY-FIRST UX PATTERNS

Rules:

- Talent contact details always masked until introduction accepted.
- Compensation is never shown; only **alignment tags** such as:
  - “Within your expectations”
  - “Above your current range”
  - “Below your current level”
- Brand sees only **public Talent profile** unless Talent engages.
- Assessment answers are never visible to Brands (only capability summaries).

---

9. TONE & MICROCOPY

Voice:

- editorial, calm, confident
- short sentences
- no superlatives
- no emoji

- no HR cliché

Common Phrases:

- “Your profile at a glance.”
- “Your next steps.”
- “Set your direction.”
- “Based on your experience.”
- “Matches aligned with your path.”
- “Discreet introduction requested.”

---

10. ACCESSIBILITY GUIDELINES

- minimum 4.5:1 contrast
- keyboard operability
- no motion dependence
- semantic HTML
- alt text on all brand imagery
- large tap zones (48px min)

---

11. UX FOR INTELLIGENCE MODULES

Matching Engine

- clean match breakdown
- badge for “High match” (>75%)

- gold-tinted accents for strong alignment

Assessment Engine

- radar chart (monochrome lines, gold highlight)
- balanced spacing

Career Projection

- timeline view (simple vertical)
- focus on ranges, not absolutes

Learning Engine

- module cards
- estimated time
- focus tags

CHAPTER 6 — COPYWRITING SYSTEM (V5 CANON)

The editorial voice of Tailor Shift.
Precise. Neutral. Confident. Discreet. Luxury-grade.

---

0. Purpose — Why Tailor Shift Needs a Copy System

Luxury retail is a culture of:

- precision,
- courtesy,

- composure,
- clarity.

Tailor Shift must speak that language.

The copy system ensures:

- every screen feels consistent,
- every word serves a purpose,
- the brand is credible to both Talents and high-end Houses,
- communication is respectful and discreet,
- the interface never feels loud, needy, or promotional.

Words are part of the product’s identity.

---

1. VOICE PRINCIPLES

Tailor Shift’s editorial voice is defined by five pillars:

1.1 Calm Confidence

- Neutral tone, stable rhythm.
- No exclamation points.
- No hype language (“amazing”, “incredible”, “game-changing”).

1.2 Precision

- Short, meaningful sentences.
- Focus on clarity, not persuasion.
- Avoid ambiguity.

- Use the correct luxury retail vocabulary (role levels, divisions, tiers, etc.)

1.3 Discretion

- Never reveal personal data prematurely.
- No pressure-driven copy (“Apply now!”).
- Respectful distance.

1.4 Professional Warmth

- Friendly, but not casual.
- Human, but not emotional.
- Always constructive.

1.5 Editorial Elegance

- Controlled use of serif for key headings.
- Thoughtful structure.
- Minimal clutter.

---

2. TONE: HOW TAILOR SHIFT SPEAKS

2.1 Do

- Clarity: “Your next step.”
- Professional respect: “You may consider this role.”
- Editorial direction: “Set your preferences.”
- Neutral encouragement: “You’re making progress.”

2.2 Don’t

- No clichés: “Dream job”, “Unlock your potential”.
- No motivational language.
- No startup tone (“Supercharge your career!”).
- No emojis.
- No slang.

---

3. GRAMMAR & STYLE RULES

3.1 Person & Address

- Use **second person** (“you”) sparingly; prefer neutral constructions.
- Avoid imperative unless guiding (“Continue”, “Save changes”).

3.2 Sentences

- Max 14–16 words for UI text.
- Avoid long subordinate clauses.
- Use active voice.

3.3 Capitalisation

- Sentence case for UI components.
- Title case for page headings.
- No all-caps except labels.

3.4 Numbers

- Always clear:

    - “3–5 years”,
    - “Level 4”,
    - “Tier 3”.

3.5 Data & Metrics

- Never exaggerate.
- Use ranges when relevant.
- Use approximations carefully (“typically”, “most profiles”, “commonly seen”).

---

4. CORE VOCABULARY (CANON)

Words the product relies on, used consistently.

Identity Vocabulary

- Profile
- Experience Block
- Published Profile
- Preferences
- Role Level (L1–L8)
- Division
- Store Type
- Complexity Tier

Matching Vocabulary

- Alignment

- Match Score
- Fit Indicators
- Role Fit
- Division Fit
- Capability Fit
- Geography Fit
- Experience Fit
- Preference Match

Career Vocabulary

- Career Route
- Projection
- Suggested Next Steps
- Focus Areas
- Recommended Learning
- Capability Summary
- Assessment

Brand Vocabulary

- House
- Employer Page
- Opportunity
- Requirements
- Compensation Band
- Store Network

- Talent Overview

---

5. COPY PATTERNS

Below are the standardised text blocks used across Tailor Shift.

5.1 Onboarding & Authentication

Headings

- “Welcome to Tailor Shift.”
- “Create your account.”
- “Tell us who you are.”
- “Set your direction.”

Body

- “This information shapes your experience.”
- “Your preferences help refine your opportunities.”
- “You can update these at any time.”

Buttons

- “Continue”
- “Save and continue”
- “Back”
- “Submit”

Microcopy

- “Your data is always private.”
- “Fields marked *required*.”

5.2 Talent Dashboard

Section titles

- “Your Profile”
- “Career Intelligence”
- “Recommended Opportunities”
- “Suggested Learning”
- “Assessment Summary”
- “Your Preferences”

Micro-insights

- “Your experience aligns with Level 3–4 roles.”
- “You may explore opportunities in similar markets.”
- “You show strong traction in client-facing roles.”

Empty states

- “Your profile is nearly complete.”
- “Add experience blocks to improve your matches.”
- “No new recommendations for now.”

5.3 Assessment (Retail Excellence Scan)

Start screen

- “This short assessment offers a perspective on your strengths.”
- “It helps personalise your experience.”

Questions

- “Which situation describes you best?”
- “How do you usually approach this?”

Results

- “Your capability overview.”
- “This reflects your experience and profile.”

Micro-insights

- “You demonstrate strong service discipline.”
- “Clienteling signals are emerging.”
- “Operational consistency is a notable strength.”

5.4 Opportunity Cards & Detail Pages

Headings

- “Role Overview”
- “Requirements”
- “Store Context”
- “Fit Indicators”

CTA

- “Request more information”
- “Express interest discreetly”
- “Save this opportunity”

Fit microcopy

- “Your experience aligns strongly with this role.”
- “Your profile is a partial match.”
- “This role may require additional experience.”

Compensation alignment (never showing Talent data)

- “Within your expected range.”
- “Above your current level.”
- “Below your current level.”

5.5 Brand Dashboard

Section titles

- “Talent Matches”
- “Recent Applicants”
- “Your Opportunities”
- “Store Network”

Actions

- “Request introduction”
- “Review profile”

- “Publish opportunity”

Microcopy

- “Contact details will be available if the Talent accepts.”
- “Matches update automatically.”

5.6 Interaction Journey (Handshake)

Talent side

- “A House has requested an introduction.”
- “Review the opportunity first.”
- “You control what is shared.”

Brand side

- “Introduction pending.”
- “You will be notified once the Talent responds.”

Acceptance

- “You are now connected.”
- “Contact details are available for both parties.”

---

6. ERROR STATES, EMPTY STATES, CONFIRMATIONS

Errors

Always neutral:

- “Something went wrong. Please try again.”
- “This action is not permitted.”
- “We couldn’t save your changes.”

Empty states

- “No data available yet.”
- “You haven’t added any experience blocks.”
- “No opportunities match your current preferences.”

Confirmations

- “Your changes have been saved.”
- “Your profile has been updated.”
- “Your request has been submitted.”

---

7. EMAIL COPY GUIDELINES

Emails follow the same tone:

Subject lines

- “Your Tailor Shift account”
- “Request received”
- “Introduction update”

Body

- short, structured, calm

- no marketing tone
- clear next steps

Signature

```
Tailor Shift
Luxury Retail Talent Intelligence
```

CHAPTER 7 — INTELLIGENCE LAYER (FOUNDATIONAL DESIGN, V5 CANON)

How Tailor Shift thinks, computes, reasons, and explains.
(Maison-agnostic, deterministic, interpretable, privacy-first.)

---

0. Purpose — What the Intelligence Layer Is

The Intelligence Layer transforms the raw data of luxury retail into:

- structured match scores,
- capability assessments,
- learning recommendations,
- route projections,
- personalised insights,
- interpretable signals for both Talents and Houses.

It is **not** a “black-box AI”.
It is a **governed, explainable reasoning stack** grounded in:

- the Master Classification System (Ch. 1),

- the Domain Entities (Ch. 2),
- the User Journeys (Ch. 3),
- the UX/UI system (Ch. 5),
- and privacy-by-design rules.

All engines are deterministic, auditable, and built to comply with the EU AI Act as a **low-risk reasoning stack**.

---

1. Principles of the Intelligence Layer

The Intelligence Layer follows eight non-negotiable principles:

1.1 Canon-First Reasoning

Every computation is based on the Master Classification System:

- Role Ladder,
- Divisions,
- Store Types,
- Complexity Tiers,
- Experience Taxonomy,
- 7-Capability Model.

Nothing in the engines bypasses or overrides the canon.

1.2 Deterministic, Explainable Outputs

No black-box logic.

Every output must be:

- deterministic (same input → same output),
- explainable (user can understand why),
- auditable (admin can inspect),
- versioned (each engine must declare its version).

1.3 Signal Aggregation, Not AI “Prediction”

The engines **combine structured signals** rather than invent predictions.

Example:

- Experience Blocks
- Role Level
- Store Context
- Divisions
- Capability Summary
- Geography
- Preferences
- Compensation Alignment (NEW)

The system interprets, organises, and ranks signals —
it does not hallucinate.

1.4 Modularity

The Intelligence Layer consists of **four core engines**:

1. **Matching Engine**
2. **Assessment Engine**

3. **Learning Engine**
4. **Career Projection Engine**

Each engine works independently, but all share:

- the same Talent entity,
- the same Opportunity entity,
- the same Capability zones,
- the same behavioural taxonomy.

1.5 Isolation of Sensitive Data

Certain data is **never used directly**, only as derived signals:

- Talent compensation
- assessment raw answers
- Talent contact information

Engines may receive:

- abstract signals (e.g., “compensation alignment: moderate”),
- filtered data (public view),
- anonymised fitness vectors.

1.6 Versioning & Governance

Each engine must:

- declare a version (e.g., `MatchingEngine_v1_2026Q1`),
- have a documented calculation model,

- be tested against a dedicated seed dataset,
- be approved before rollout.

1.7 Low-Latency, High-Accuracy

The intelligence layer serves luxury retail:
the experience must be smooth, quick, and respectful.

Targets:

- p50 < 250 ms for matching a single Talent ↔ Opportunity
- p50 < 150 ms for assessment scoring
- p50 < 300 ms for projection

1.8 No Client-Side Intelligence

Nothing is computed client-side.

All engines run:

- server-side functions
- or Tiger Cloud server actions
- or scheduled processes
- or fully controlled Edge Functions

This ensures privacy and integrity.

---

2. Architecture of the Intelligence Layer

The Intelligence Layer is structured into **4 horizontal modules** and **1 vertical orchestrator**.

2.1 Horizontal Modules (The Engines)

1. Matching Engine (Ch. 8)

Purpose:
Rank how well a Talent aligns with an Opportunity.

Inputs:

- Talent fields (role level, divisions, experience, preferences, geography)
- Opportunity fields (requirements, store context, compensation band)
- Derived signals (compFit, capability summary)

Outputs:

- global match score
- 7D breakdown
- alignment tags

---

2. Assessment Engine (Ch. 9)

Purpose:
Generate a behavioural capability snapshot (Service, Clienteling, Leadership, Operations).

Inputs:

- assessment answers
- experience blocks
- role level
- division and tier exposure

Outputs:

- capability scores (0–100)
- capability summary
- insights

---

3. Learning Engine (Ch. 10)

Purpose:
Recommend modules based on gaps and strengths.

Inputs:

- capability summary
- experience blocks
- role level
- preferences
- match weaknesses

Outputs:

- ranked list of learning modules
- focus areas

---

4. Career Projection Engine (Ch. 11)

Purpose:
Suggest realistic next role levels, store tiers, divisions, and geographies.

Inputs:

- role level
- capability summary
- division experience

- tier exposure
- mobility level
- preferences

Outputs:

- projected next role
- projected next store tier
- suggested hubs
- timeline ranges
- gap analysis

2.2 Vertical Module (The Orchestrator)

A fifth layer ties everything together:

The Intelligence Orchestrator

Purpose:
Coordinate engines for each Talent.

Responsibilities:

- trigger matching after profile updates
- recompute projections after assessment
- refresh recommendations after learning updates
- update dashboard summaries
- queue computations when needed
- produce versioned analytic logs

The Orchestrator never computes; it orchestrates.

3. Data Flow Overview

Below is the canonical flow across engines.

3.1 Talent → Assessment → Projection

1. Talent updates Experience Blocks
2. Orchestrator triggers Assessment Engine
3. Capability summary updates
4. Projection Engine re-runs
5. Dashboard updates:
    - readiness
    - focus areas
    - next role, next tier

3.2 Opportunity → Matching

1. House posts Opportunity
2. Orchestrator:
    - prepares Opportunity vector
    - runs Matching Engine for relevant Talents
3. Brand Dashboard shows ranked list
4. Talent Dashboard surfaces aligned roles

3.3 Learning Feedback Loop

1. Talent completes module
2. Learning Engine updates gap status
3. Projection Engine adjusts readiness
4. Dashboard updates

3.4 Compensation Alignment (NEW)

Sensitive Talent compensation is never shown.

Instead:

1. Talent enters compensation data
2. Opportunity enters compensation band
3. Engine computes:
   - **compFit** (0–100)
   - alignment tag
4. compFit influences Preference Fit
5. No raw figures are revealed to either side

---

4. Privacy-by-Design Rules for Engines

4.1 No direct access to:

- Talent compensation
- assessment answers
- email or phone

- interaction history

4.2 Only derived signals are passed:

- capability_summary
- compFit
- alignment tags
- geography_fit
- experience_fit

4.3 House-facing views are filtered:

- Talent public profile only
- No sensitive data
- Only what Talent has consented to show

---

5. Intelligence Outputs — What Users See

Talent Sees:

- their capability summary
- match breakdown for each opportunity
- recommended learning modules
- projection insights
- preference alignment

Brand Sees:

- Talent public card
- match breakdown (filtered)
- experience blocks (condensed)
- role level, divisions, assessment summary
- “introduction pending” state

Both See:

- contact details **only** after mutual acceptance

---

6. Engine Versioning System

Every engine must embed:

```
version: “MatchingEngine_v1_2026_Q1”
calculation_schema: JSON
weights: JSON
explainability_reference: URL
```

Versioning ensures:

- reproducibility
- auditability
- regulatory compliance
- clear change logs

---

7. Implementation Requirements

7.1 Engine Code Must:

- be pure functions
- accept typed inputs
- return typed outputs
- throw structured errors
- be testable independently

7.2 Engines Must Produce:

- deterministic objects
- explainability metadata
- logs (non-sensitive)
- cache-friendly outputs

CHAPTER 8 — MATCHING ENGINE (V5 CANON)

The unified 7-Dimensional matching system for Talent ↔ Opportunity.
Deterministic. Explainable. Private. Luxury-specific.

---

0. Purpose — What the Matching Engine Does

The Matching Engine:

- evaluates **how well a Talent aligns** with a given Opportunity,
- produces a **global match score** (0–100),
- generates a **7D breakdown** (per dimension),
- computes **alignment tags** for UX (“Strong Alignment”, “Partial Fit”, “Below Requirements”),

- ensures Talent privacy (no sensitive data ever revealed),
- ranks opportunities for Talents,
- ranks Talents for Houses,
- supports projection, learning, and Talent discovery.

The engine is **not** a vague algorithm.
It is a **rules-based, auditable system** grounded in the luxury retail ontology.

---

1. Architecture of the Matching Engine

The engine evaluates **7 dimensions**:

1. **Role Fit**
2. **Division Fit**
3. **Store Tier / Store Context Fit**
4. **Capability Fit**
5. **Geography Fit**
6. **Experience Block Fit**
7. **Preference Alignment** *(including compensation alignment)*

Each returns a 0–100 score.

The **global score** is a weighted average of the 7 dimensions.

---

2. The 7 Dimensions (Detailed)

Below is the deterministic logic for each dimension.

2.1 ROLE FIT (Weight: HIGH)

Measures how well a Talent's role level aligns with the opportunity's normalized role level.

Inputs:

- `talent.current_role_level`
- `opportunity.normalized_role_level`

Rules:

Difference	Score
0 (same level)	100
±1	85
±2	55
±3	25
±4 or more	0

Interpretation:

- Adjacent levels are encouraged (natural progression).
- Large jumps are strongly discouraged unless explicitly allowed by preferences.

2.2 DIVISION FIT (Weight: HIGH)

Matches Talent product specialization to Opportunity needs.

Inputs:

- `talent.divisions_experience[]`

- `opportunity.divisions_required[]`

Rules:

- Intersection = strong signal
- No intersection = weak or zero

Scoring:

`intersection_ratio = |intersection| / |required_divisions|`

- `1.0` → 100
- `0.66` → 80
- `0.33` → 50
- `0` → 0

Notes:

- High Jewelry, Watches, and Beauty roles have stricter expectations.
- Multi-division stores apply weighted logic.

2.3 STORE TIER / STORE CONTEXT FIT (Weight: MEDIUM–HIGH)

Luxury stores differ drastically in complexity.

Inputs:

- Talent's historical store exposure (t1–t5)
- Opportunity store context

- Store type

Logic:

- If Talent has experience at **same tier** → strong
- If Talent has **tier -1 or +1** → acceptable
- Jumps >2 tiers → weak
- Jumps >3 tiers → 0

Scoring:

Tier Difference	Score
0	100
1	80
2	55
3	20
≥4	0

Store Type Adjustment (±10):

- Specialty boutiques → neutral
- High Jewelry salons → require validated experience
- Travel Retail → +10 for fast-paced profiles
- Flagships → require Leadership + Ops signals

2.4 CAPABILITY FIT (Weight: MEDIUM–HIGH)

Based on the Assessment Engine’s capability summary.

Inputs:

- `talent.assessment_summary` (service, clienteling, operations, leadership signals)
- `opportunity.min_assessment_scores` (optional)

Calculated via:

```
per-dimension-gap = max(0, expected - actual)
score = 100 - (average_gap * factor)
```

If no assessment:

- Use an **inferred baseline** from role level + experience blocks
- Maximum score capped at 70 (to encourage taking assessment)

2.5 GEOGRAPHY FIT (Weight: MEDIUM)

Aligns Talent mobility with Opportunity location.

Inputs:

- Talent location
- Talent mobility level
- Opportunity location
- Preferences

Rules:

Condition	Score
same city	100
same region	80

same country	70
region-only and matches	60
global_ok	40
out of scope	0

2.6 EXPERIENCE BLOCK FIT (Weight: MEDIUM)

Evaluates alignment between required behavioural patterns and block history.

Inputs:

- FOH / BOH / Leadership / Clienteling / Ops / Business distribution
- Achievements
- Years in block

Logic:

- Map block types to opportunity expectations
- Leadership-heavy roles → heavy penalty if no Leadership blocks
- VIC-heavy roles → require Clienteling blocks
- Ops-heavy → require Operations + BOH blocks

Scoring:

```
match_ratio = aligned_blocks / expected_blocks
```

Additional heuristics:

- 2+ years in category = stability bonus

- achievements list length = signal strength

2.7 PREFERENCE ALIGNMENT (Weight: MEDIUM–LOW)

Aligns Talent stated preferences with Opportunity attributes.

Subcomponents:

- Target role levels
- Target divisions
- Target store types
- Target regions/cities
- Target brands
- Compensation Alignment (NEW)

Each contributes independently.

---

3. COMPENSATION ALIGNMENT (NEW, V5)

Integrated into Preference Alignment. Never reveals Talent data.

Inputs:

- Talent’s confidential compensation profile
- Opportunity’s public compensation band
- Derived Talent total compensation (base + variable + allowance)

- Derived Opportunity midpoint compensation

Logic:

Compute relative difference:

```
ratio = talent_total / opportunity_mid
diff = abs(ratio - 1)
```

Scoring:

Difference	Score
≤ 0.25	100
≤ 0.50	80
≤ 1.00	60
> 1.00	40

Output to UX:

Instead of numbers → alignment tags:

- "Within your expected range"
- "Above your current level"
- "Below your current level"

Privacy Guarantee:

Brands **never** see:

- Talent salary
- Talent expectations
- any raw compensation data

---

4. GLOBAL MATCH SCORE

The global score is a weighted sum:

```
global_score =
  0.20 * RoleFit +
  0.20 * DivisionFit +
  0.15 * StoreTierFit +
  0.15 * CapabilityFit +
  0.10 * GeographyFit +
  0.10 * ExperienceBlockFit +
  0.10 * PreferenceAlignment
```

Weights are adjustable in future versions but fixed for V5.

---

5. EXPLAINABILITY MODEL (for UX)

Each dimension produces:

- score (0–100),
- short explanation sentence,
- "Fit Tag" (Strong, Medium, Partial, Weak).

Examples:

Role Fit:

> "Your role level is closely aligned with the expectations for this position."

Division Fit:

> "You have strong experience in the required product divisions."

Capability Fit:

> "Your assessment indicates solid service and operations signals."

Compensation Alignment:

“This role is within your expected compensation range.”

All explanations are **data-neutral** and **calm**.

---

6. SCORING PIPELINE (Orchestrator)

1. Prepare Talent vector
2. Prepare Opportunity vector
3. Compute 7D scores
4. Combine into global score
5. Generate explainability bundle
6. Persist match snapshot (optional)
7. Rank results

---

7. PERFORMANCE & LATENCY TARGETS

- Matching a single Talent ↔ Opportunity pair: **< 250ms**
- Matching one Talent against 20 opportunities: **< 400ms**
- Matching one opportunity against 100 Talents: **< 800ms**

Caching used for:

- Talent capability summary
- Experience block vectorisation
- Opportunity requirement signatures

8. IMPLEMENTATION REQUIREMENTS

Engine code:

- pure functions
- fully typed
- no network calls
- no side effects

Data:

- no sensitive data inside matching logs
- no raw compensation data outside engine scope

Testing:

- 20+ matching scenarios in integration suite
- “explainability test” per dimension
- regression suite for changes in weighting

CHAPTER 9 — ASSESSMENT ENGINE (V5 CANON)

Measuring luxury retail capabilities with precision, respect, and explainability.

---

0. Purpose — What the Assessment Engine Is

The Assessment Engine produces a **capability signature** for each Talent.

It converts behavioural and experiential signals into:

- **capability scores** (0–100),
- **insights**,
- **gap analysis**,
- **readiness indicators**,
- **inputs for the Matching Engine**,
- **inputs for the Projection Engine**,
- **inputs for the Learning Engine**.

It is:

- deterministic,
- maison-agnostic,
- grounded in the MCS capability model,
- privacy-preserving,
- explainable,
- low-risk by design (EU AI Act compliant).

No deep learning.
No opaque scoring.
No psychometrics.
Only **structured behavioural interpretation**.

---

1. Inputs — What the Engine Consumes

The Assessment Engine does **not** require all fields to function, but it improves with more data.

Mandatory inputs

- answers to the Retail Excellence Scan (10–12 structured questions)

Optional / enhancing inputs

- Talent’s role level (L1–L8)
- Experience Blocks (FOH, Leadership, etc.)
- Division exposure
- Store tier history
- Self-declared mastery levels
- Years of experience

Never used

(Sensitive or out of scope by design)

- compensation
- contact details
- raw preferences
- mobility
- demographic attributes
- subjective free-text that cannot be interpreted deterministically

---

2. Outputs — What the Engine Produces

The Assessment Engine outputs:

2.1 Capability Scores (0–100)

Four dimensions in V5:

1. **Service Excellence**

2. **Clienteling & Relationship Building**
3. **Operational Reliability**
4. **Leadership Signals**

2.2 Capability Summary (short-form)

A compact vector:

```
{
  service_excellence: number,
  clienteling: number,
  operations: number,
  leadership_signals: number
}
```

Stored in:

- `Talent.assessment_summary`
- also duplicated into Assessment record

2.3 Insight List

Short, editorial insights such as:

- “Your operational discipline is consistently strong.”
- “Clienteling signals are present and developing.”
- “You show early leadership behaviours.”

2.4 Gap Indicators

Indicates which dimensions fall below a threshold (default: <60).

2.5 Readiness Indicators

Used by the Projection engine to estimate:

- “readiness for next role level”,

- “readiness for more complex stores”.

---

3. Retail Excellence Model — Capability Definitions (V5)

The Assessment Engine relies on the **4D capability model** for V1–V2 (Lite Scan).
The full 7D model appears in V3.

3.1 Service Excellence

Measures:

- consistency of selling ceremony steps
- attentive behaviour
- pace and flow control
- ability to adapt to customer needs
- product knowledge basics
- attention to detail

3.2 Clienteling & Relationship Building

Measures:

- proactivity in follow-ups
- segmentation discipline
- confidence in VIP / VIC interactions
- long-term relationship mindset
- CRM fluency

3.3 Operational Reliability

Measures:

- accuracy in BOH tasks
- compliance discipline
- cash/stock reliability
- process consistency
- attention to standards & routines

3.4 Leadership Signals

Measures:

- leading small shifts
- supporting peers
- pacing and coordination
- handling floor tension
- coaching signals
- initiative-taking

---

4. Assessment Flow (UX Journey)

(Defined in detail in Ch. 3; repeated here from an engine perspective.)

1. Talent begins the scan
2. Responds to 10–12 scenario-based questions
3. Responses encoded into discrete behavioural tags
4. Engine interprets tags into capability contributions
5. Weighted scoring is applied
6. Insight templates generated

7. Score stored and displayed

No branching logic.
No non-deterministic behaviour.

---

5. Question Architecture (V5)

The scan uses:

5.1 Scenario-based behavioural questions

Examples:

- “When the store is busy, you usually…”
- “With returning clients, you typically…”
- “When a task requires precision, you…”
- “During moments of tension, you…”

5.2 Four answer options per question

Each aligned with one or multiple capabilities.

5.3 Scoring Model

Each option contributes:

- +X to service
- +Y to operations
- +Z to clienteling
- +W to leadership

Example (not exhaustive):

Option	Service	Clienteling	Ops	Leadership

A	8	1	4	0
B	5	6	1	2
C	2	3	8	1
D	0	0	0	8

5.4 Normalisation

After all answers:

```
raw_score[d] = sum(contributions)
normalised[d] = (raw_score[d] / max_possible[d]) * 100
```

---

6. Enhanced Scoring (Contextual Adjustments)

The Assessment Engine does not operate in isolation.
It incorporates contextual data to avoid misleading results.

6.1 Role Level Adjustment

Higher role levels require higher capability maturity.
Scaled expectations:

- L1–L2: lenient
- L3–L4: moderate
- L5+: demanding

Adjustment:

```
adjusted_score = base_score - (role_multiplier * role_gap)
```

6.2 Experience Block Adjustment

Based on:

- FOH / Clienteling exposure
- Leadership exposure
- Ops / BOH exposure

Example:

- Talent with multiple Clienteling blocks gets a floor (min) score for clienteling dimension.

6.3 Division Adjustment

Certain divisions (High Jewelry, Watches, Beauty) require higher standards in:

- storytelling
- technical knowledge
- precision
- pace

Minor weighting adjustments apply.

6.4 Store Tier Adjustment

Talent exposed to T3–T1 environments may have:

- higher expectations
- stronger operational baseline

Adjustment applies downward or upward based on achieved tier.

---

7. Insight Engine — How Insights Are Generated

Based on the 4 dimension scores plus key behavioural markers.

Format:

Strong Signals (>75)

- “Your service consistency stands out across your profile.”

Growing Signals (60–75)

- “You show solid foundations in operational routines.”

Developing Signals (40–60)

- “You may consider strengthening your clienteling routines.”

Weak Signals (<40)

- “Operational reliability is an area of opportunity for you.”

Insights are:

- short
- neutral
- non-evaluative
- forward-looking
- editorial (not algorithmic-sounding)

---

8. Integration Into Other Engines

8.1 Matching Engine

Capability Fit is derived **directly** from capability scores.

8.2 Learning Engine

Learning suggestions target weak or emerging capabilities.

8.3 Projection Engine

Capability maturity influences:

- next role level
- store tier readiness
- hub suitability
- timeline

8.4 UX

Assessment results:

- displayed in Talent dashboard,
- simplified in Talent profile,
- abstracted (never raw answers) in Brand-facing view,
- used for internal readiness logic.

---

9. Privacy & Compliance Rules

9.1 Never stored in logs

- raw answers
- sensitive behavioural flags

9.2 Never shared with Houses

- answers
- low-level capability data

Brands only see:

- high-level capability summary (e.g., “Service: Strong”)
- only AFTER Talent approves an introduction

9.3 Data Minimisation

Keep only:

- scores
- insights
- version
- timestamp

Discard everything else.

---

10. Versioning & Testing

10.1 Version ID

Example:

`AssessmentEngine_v1_2026_Q1`

10.2 Regression Tests

- same input → same output
- test across 20–30 synthetic Talent profiles
- ensure no drift after copy or weighting changes
- verify insight coherence

10.3 Integrity Tests

- input validation

- missing fields gracefully handled
- clear error messages

CHAPTER 10 — LEARNING ENGINE (V5 CANON)

A deterministic, capability-driven recommendation engine for luxury retail talent development.

---

0. Purpose — What the Learning Engine Is

The Learning Engine converts a Talent's:

- **capability scores**,
- **experience blocks**,
- **role level**,
- **preferred trajectory**,
- **divisions**,
- **store tier history**,
- **projection outputs**,

into **highly targeted learning recommendations**.

It is *not* a general e-learning system.
It is a **precision coaching engine** aligned with the realities of luxury retail.

Goals:

- strengthen capability gaps,

- reinforce existing strengths,
- prepare for next role level / store tier,
- guide talent progression with discretion and clarity.

---

1. Inputs — What the Engine Consumes

The Learning Engine ingests the following canonical signals:

1.1 Capability Summary (mandatory)

- service_excellence
- clienteling
- operations
- leadership_signals

(from Assessment Engine)

1.2 Experience Blocks

- FOH / BOH / Leadership / Clienteling / Ops / Business
- achievements
- years per block
- division exposure
- tier exposure

1.3 Role Level

L1–L8 expectations shape recommended modules.

1.4 Career Preferences

Target:

- divisions
- role levels
- store types
- regions
- brands/groups

1.5 Projection Engine Output

- next role level
- next store tier
- readiness gaps
- recommended focus areas

1.6 Learning History

- completed modules
- badges
- progress metrics

---

2. Outputs — What the Engine Produces

The Learning Engine returns a ranked list of recommendations:

```
{
  recommendations: [
    {
      module_id: UUID,
      title: string,
      category: string,
      duration_minutes: number,
      rationale: string,
      priority: 'high' | 'medium' | 'low'
    },
    ...
  ]
}
```

Plus:

- **Focus Areas:**
  "Strengthen clienteling routines."
  "Improve operational consistency."

- **Short Insights:**
  "This module aligns with your target role level."
  "You may benefit from reinforcing leadership behaviours."

- **Readiness Progression:**
  "Completing these modules brings you closer to Level 4."

---

3. Learning Module Taxonomy (V5)

Learning Modules exist in six primary categories:

1. **Service Excellence**
2. **Clienteling & CRM**
3. **Operations & Compliance**
4. **Leadership & People Development**
5. **Brand Ambassadorship**
6. **Business Performance**

Each module contains:

- title
- summary

- time to complete
- content type (video, text, slides)
- capability mapping
- recommended role levels
- recommended divisions
- recommended store tiers

---

4. Recommendation Logic — Deterministic Rules

The engine uses a **three-layer logic stack**:

1. **Gap-based Matching**
2. **Trajectory-based Recommendations**
3. **Contextual Refinements**

4.1 Layer 1 — Gap-Based Matching (Primary Driver)

Identify weaknesses and suggest modules to strengthen them.

Algorithm:

```
for each capability in [service, clienteling, operations,
leadership]:
    if capability_score < 60:
        gap_severity = (60 - capability_score)
        recommend related modules
```

Mapping:

- Service gaps → service excellence modules
- Clienteling gaps → CRM & clienteling modules
- Ops gaps → BOH / operations modules
- Leadership gaps → people development modules

Priority:

- <40 → high
- 40–60 → medium
- 60 → low

4.2 Layer 2 — Trajectory-Based Recommendations

Using Projection Engine outputs:

If next role level requires:

- **more leadership** → recommend leadership modules
- **more tier exposure** → recommend operations/complexity modules
- **more clienteling** → focus on CRM/VIC modules
- **more service mastery** → service excellence modules

Example:

Talent Level 3 projected to Level 4:

- needs stronger team pacing → Leadership L3–L4 content

- needs operational consistency → Operations L2–L4
- needs clienteling confidence → Clienteling L2–L4

4.3 Layer 3 — Contextual Refinements

Contextual adjustments refine the final recommendation list.

3.1 Division Context

Watches/HJ → advanced product knowledge
Beauty → storytelling, routine guidance
RTW → styling, silhouette logic
LG → craft literacy

3.2 Store Tier Context

High-tier stores require:

- elevated service rituals
- stronger coordination skills
- complex operations mastery

3.3 Experience Blocks Context

If Talent has no Leadership blocks but is projected for L4 → leadership modules receive **high** priority.

3.4 Preferences Context

If Talent prefers:

- High Jewelry → jewel storytelling modules
- Travel Retail → pace & flow modules
- Flagships → multi-division coordination modules

3.5 Learning History

Modules already completed → deprioritised
Modules in-progress → medium priority

---

5. Ranking Algorithm — Final Output

After applying all layers, modules are ranked by:

1. **Gap severity**
2. **Trajectory relevance**
3. **Division & tier context**
4. **Recency of last learning**
5. **Completion status**

Scoring:

```
score = w1*gap_score + w2*trajectory_score + w3*context_score -
w4*completed_penalty
```

Weights:

- gap (w1) = 0.45
- trajectory (w2) = 0.30
- context (w3) = 0.20
- penalty (w4) = 0.40

Final list sorted descending.

---

6. Example Recommendation Set

For Talent:

- Level 3 aiming for Level 4
- clienteling score: 52
- operations score: 64
- leadership score: 48

Recommendations:

1. **“Leading Effective Store Shifts”**
   - Priority: High
   - Rationale: Leadership signals below Level 4 expectation
   - Duration: 15 min
2. **“Clienteling Foundations for Tier 2–3 Boutiques”**
   - Priority: Medium
   - Rationale: Clienteling score below threshold
   - Duration: 10 min
3. **“Operational Discipline: Daily Routines”**
   - Priority: Low
   - Rationale: Reinforce tier expectations
   - Duration: 12 min

---

7. UX Presentation — How Learning Appears in the App

Talent Dashboard:

- “Suggested Learning” card with 2–3 modules

- gold accent on high-priority modules
- simple tags (“Leadership”, “Clienteling”, “Operations”)

Learning Page:

- full list sorted by priority
- category filters
- completed modules separated

Module Detail:

- title
- summary
- category
- duration
- rationale (“Recommended based on your capability profile.”)

---

8. Privacy & Compliance

8.1 Raw assessment answers never used

Only capability summary enters the Learning Engine.

8.2 No inference from compensation

Learning is unrelated to pay.

8.3 House access

Brands **never** see:

- module history

- learning performance
- scores or gaps

8.4 Data retention

Minimal storage: completion timestamps only.

---

9. Versioning & Testing

Version Format

`LearningEngine_v1_2026_Q1`

Unit Tests

- correct ranking logic
- correct module filtering
- consistent interpretation

Integration Tests

- Assessment ↔ Learning
- Projection ↔ Learning

CHAPTER 11 — CAREER PROJECTION ENGINE (V5 CANON)

Predictive career pathways for luxury retail talent, grounded in ontology rather than speculation.

---

0. Purpose — Why a Projection Engine?

Luxury retail careers follow **recognisable, structured patterns**.
The Projection Engine identifies these patterns and produces:

- the **next likely role level**,
- the **next plausible store tier**,
- **division trajectories**,
- **geographic pathways**,
- **timelines** (ranges, never promises),
- **capability gaps to bridge**,
- **recommendations** to accelerate progression.

It is:

- deterministic,
- maison-agnostic,
- interpretable,
- low-risk,
- built from real industry logic,
- not a machine-learning black box.

---

1. Inputs — What the Engine Consumes

The Projection Engine synthesises:

1.1 Core Talent Profile

- current role level (L1–L8)
- years of experience
- divisions (experience & mastery)

- store tier exposure
- experience blocks
- mobility level
- geography

1.2 Assessment Summary

4D capability scores:

- service
- clienteling
- operations
- leadership

1.3 Preferences

- target role levels
- target divisions
- target store types
- target regions
- target brands

1.4 Learning Status

- completed modules
- capability gaps
- readiness signals

1.5 MCS Constraints

- role ladder

- tier progression logic
- division adjacency
- geography clusters

---

2. Outputs — What the Engine Produces

The Projection Engine returns a structured projection object:

```
{
  next_role_level: number,
  next_store_tier: complexity_tier_enum,
  projected_hubs: string[],
  role_timeline_range: [min_months, max_months],
  tier_readiness: 'strong'|'medium'|'developing'|'weak',
  capability_gaps: string[],
  recommended_focus: string[],
  division_trajectory: string[],
  notes: string
}
```

Additionally, it provides:

- **Projected arcs** (“L3 → L4”, “T2 → T3”, “Fashion + Leather Goods trajectory”)
- **Narrative insights** for the Talent dashboard

---

3. Core Logic — How Projection Works

The engine runs through **five layers of reasoning**:

1. **Role Level Progression**
2. **Store Complexity Projection**

3. **Division Trajectory**
4. **Geographic Pathways**
5. **Timeline Estimation**

Each layer is deterministic and governed by the MCS.

---

4. Layer 1 — Role Level Progression (L1→L8)

The engine determines whether a Talent is ready for the next role level.

4.1 Determinants

- capability maturity
- leadership signals
- FOH/Operations balance
- years in-role
- store tier exposure
- experience blocks distribution
- division complexity
- learning history

4.2 Canonical Thresholds

Current Level	Typical Next Step	Requirements
L1 → L2	Senior Advisor	Advanced service + early clienteling
L2 → L3	Team Leader	Emerging leadership + ops discipline
L3 → L4	ASM	solid leadership + multi-division awareness

L4 → L5	Store Manager	full ops + people dev + commercial logic
L5 → L6	Flagship Director	multi-division coordination + VIC
L6 → L7	Cluster Manager	multi-boutique leadership
L7 → L8	Regional / Country	strategic capability + mobility

These are *industry-standard patterns*, not Tailor Shift inventions.

4.3 Scoring Readiness

```
role_readiness = weighted_sum(
  leadership_signals,
  service_excellence,
  clienteling,
  operations,
  tier_experience_factor,
  division_factor
)
```

Thresholds:

- 75 → Strong readiness
- 60–75 → Medium
- 45–60 → Developing
- <45 → Weak

---

5. Layer 2 — Store Complexity Projection (T5→T1)

Store tiers influence role readiness.

Inputs:

- years in each tier

- block experience
- leadership exposure
- division exposure

Canonical Tier Rules:

- Tier ↑ is a **progression** (more complexity).
- Moving from T3 → T2 → T1 indicates increasing operational maturity.
- Multi-division + clienteling-heavy = higher tier expectations.

Projection:

- If Talent L3 with strong Ops → T3/T2 recommended
- If Talent L4 with strong Leadership → T2/T1
- If strong clienteling & service → T1 (Flagship) considered

Example:

```
{
  next_store_tier: "T2",
  tier_readiness: "medium"
}
```

---

6. Layer 3 — Division Trajectory

The engine examines division patterns:

Inputs:

- divisions_experience
- division mastery self-score
- division category complexity

- preferences

Logic:

- If Talent excels in RTW + LG → dual-division trajectory → common in multi-category houses
- If Talent strong in Watches → watches trajectory
- If Talent strong in HJ → luxury specialization trajectory

Example:

```
division_trajectory: ["RTW", "Leather Goods"]
```

---

7. Layer 4 — Geographic Projection

Geography shapes career opportunities.

Inputs:

- current geography
- mobility
- house clusters (hidden logic)
- hub adjacency

Canonical Hubs:

- Paris
- Milan
- London
- Dubai
- Singapore

- Hong Kong
- Shanghai
- New York
- Tokyo

Logic:

- If mobility = “same city only” → no projection
- If mobility = “same country” → suggest hubs in that country
- If mobility = “region_only” → suggest regional hubs
- If “global_ok” → suggest hubs with strong match density

Example:

```
projected_hubs: ["Milan", "Dubai"]
```

---

8. Layer 5 — Timeline Estimation

Timeline ranges are **non-binding ranges**, based on:

- role level
- capability maturity
- tier experience
- division complexity
- learning progress

Typical patterns:

Transition	Timeline
L1 → L2	6–12 months

L2 → L3	9–18 months
L3 → L4	12–24 months
L4 → L5	18–30 months
L5 → L6	24–42 months

Output example:

```
role_timeline_range: [12, 18]
```

---

9. Gap Analysis

The Projection Engine highlights capability gaps that slow progression.

Example:

- “Strengthen leadership routines for Level 4 readiness.”
- “Enhance clienteling depth for high-tier environments.”
- “Operational reliability is essential for progression.”

This drives the Learning Engine (Ch. 10).

---

10. Integration With Other Engines

10.1 With Matching

Higher readiness → higher Role Fit score.

10.2 With Learning

Projection provides the **“why”** behind recommended modules.

10.3 With Assessment

Assessment → capability maturity → readiness.

10.4 With UX

Displayed elegantly in the dashboard:

- next role level
- next store tier
- projected hubs
- readiness tags
- focus areas

---

11. Explainability

Every projection includes:

- rationale
- sample trajectories
- capability-based reasoning
- clear disclaimers

Example explanation:

> “Based on your leadership signals, operational consistency, and previous tier exposure, you appear aligned with Level 4 responsibilities within a 12–18 month range.”

---

12. Privacy Rules

The Projection Engine:

- never uses compensation

- never uses personal demographic data
- never reveals sensitive details to Brands
- only shares high-level “capability summary” in brand-facing views
- contact details always masked
- timeline ranges never appear for Brands

---

13. Versioning & Testing

Version Format

`ProjectionEngine_v1_2026_Q1`

Regression Tests

- consistent outputs
- correct trajectory mapping
- correct timeline logic

Domain Tests

- against canonical seed data
- against luxury benchmarks

CHAPTER 12 — PERSONALISATION ENGINE (V5 CANON)

Discrete, predictable, privacy-first personalisation for luxury retail talent and houses.

---

0. Purpose — What Personalisation Means in Tailor Shift

Tailor Shift is **not** a social platform and **not** a recommender system with algorithmic guessing.
Its personalisation must be:

- **discreet** (never invasive),
- **deterministic** (no black-box),
- **transparent** (explainable reasoning),
- **privacy-preserving**,
- **lightweight**,
- **grounded in ontology-driven logic**,
- **useful**,
- **never manipulative**.

The Personalisation Engine orchestrates:

1. **What Talent sees**
2. **What Brands see**
3. **How content is ordered, filtered, and highlighted**
4. **How insights appear in dashboards**
5. **How journeys adapt to Talent maturity**
6. **How preferences impact recommendations**
7. **How compensation alignment is integrated discreetly**

It leverages outputs from:

- Matching Engine

- Assessment Engine
- Learning Engine
- Projection Engine
- user preferences

The Engine **never** uses personal or demographic data.
It operates purely on behavioural, professional, and contextual signals.

---

1. Inputs — Signals Used by the Engine

Personalisation draws from a strictly limited, safe data set.

1.1 Talent Inputs

- role level
- divisions
- store tier experience
- assessment summary
- capability gaps
- experience blocks
- preferences (career, store types, divisions, geographies, brands)
- mobility
- openness status
- learning history
- interaction history (high-level)

1.2 Opportunity Inputs

- normalized role level
- division requirements
- store tier
- geography
- compensation band
- capability expectations

1.3 Brand Inputs

- brand divisions
- store network
- opportunity list

1.4 System Signals

- match scores
- projection outputs
- learning recommendations
- dashboard context
- UX state (mobile/desktop)

Never used:

- raw compensation
- contact details
- demographic data
- psychometric data
- unverified free text

- inferred sensitive attributes

Tailor Shift stays strictly compliant with data minimisation.

---

2. Outputs — What the Engine Produces

The engine is not a single output; it generates a set of personalised surfaces:

2.1 Talent Dashboard Personalisation

- priority cards
- match-ranked opportunities
- assessment summary
- recommended learning
- projection insights
- capability highlights
- preference reminders

2.2 Opportunity Personalisation

- match-level sorting
- fit breakdown
- “why this opportunity matters” labels
- compensation alignment tag (no raw values)

2.3 Brand Dashboard Personalisation

- ranked Talent matches
- priority candidates

- fit breakdown
- search results weighted by relevance

2.4 Learning Personalisation

- sorted module recommendations
- focus areas
- module progress

2.5 Assessment Personalisation

- insights aligned with role-level expectations
- context-based gap messaging

2.6 Interaction Personalisation

- who sees what, when
- dynamic gating
- discrete introduction logic

2.7 Navigation Personalisation

- on desktop: sidebar emphasis
- on mobile: contextual CTA consolidation

---

3. Personalisation Layers (Deterministic)

The engine uses **four layers of deterministic rules**.
Each layer refines the previous one.

3.1 Layer 1 — Structural Personalisation (Based on Role & Journey Stage)

For Talent

Stage	Display Priority
New profile	onboarding cards + identity tasks
Post-onboarding	assessment prompt + profile completion
Post-assessment	learning + projections
Mature profile	opportunities + learning + projection

For Brand

- focus on opportunities first
- store network second
- talent discovery third

Structural personalisation makes sure the right “chapter” appears at the right time.

3.2 Layer 2 — Professional Personalisation (Based on Capability & Experience)

The engine maps the Talent profession to:

- opportunity recommendations,
- learning modules,
- readiness signals.

Example:

If Talent is L3 with strong operations but weak leadership:

- surface leadership-focused modules
- surface opportunities requiring L3–4 progression
- show “preparation for next level” insights

If Talent strong in clienteling:

- surface boutique roles with high VIC density
- highlight opportunities with division match

3.3 Layer 3 — Preference Personalisation (Based on Talent-Declared Intentions)

The engine applies strict preference filters:

If Talent selects:

- **target divisions** → opportunity list filtered & boosted
- **target store types** → emphasised in cards
- **target brands** → prioritised
- **mobility constraints** → filter geography

Preferences directly influence:

- what opportunities appear
- how they are ranked
- how match explanations are phrased

Talent preferences override all other logic.

3.4 Layer 4 — Contextual Personalisation (Based on Interaction & Environment)

The engine adjusts content based on:

Interaction context

- recent interest → boost opportunity
- recent brand request → highlight notification
- pending introduction → pinned card

Geography context

- local roles within 25–50 km radius boosted
- regional hubs highlighted if mobility level permits

Time context

- after completing assessment → projection surfaces
- after completing modules → refreshed suggestions

Device context

- mobile: prioritise simple, vertical cards
- desktop: priority grid

Compensation context (privacy-preserving)

- uses compFit tag
- never exposes Talent numbers
- may highlight compatible opportunities

---

4. Ranking Algorithms — How Content Is Ordered

Personalisation uses weighted rules (no ML models, no statistical prediction).

4.1 Opportunity Ranking for Talents

```
score =
  0.55 * match_score +
  0.25 * preference_alignment +
  0.10 * projection_relevance +
  0.05 * recency_of_opportunity +
  0.05 * comp_alignment_tag_weight
```

The list is then:

- filtered by geography
- filtered by excluded divisions (if any)
- sorted by score

4.2 Talent Ranking for Brands

```
score =
  0.60 * match_score +
  0.25 * assessment_strength +
  0.10 * experience_block_alignment +
  0.05 * recent_activity
```

Brand sees an ordered list but with **masked contact**.

4.3 Learning Module Ranking

```
score =
  0.45 * gap_severity +
0.30 * trajectory_relevance +
0.20 * division_context +
0.05 * novelty_factor
```

---

5. Personalised Insight Engine

Generates **short, editorial insights**, similar in tone to Chapter 6.

Examples:

Opportunities:

- “This role aligns well with your level and division experience.”
- “You may explore this opportunity due to strong operational signals.”

Capability:

- “Your leadership signals support L4 progression.”
- “Clienteling depth is emerging; consider strengthening it.”

Projection:

- “Professionals with similar profiles often progress to Level 4 within 12–18 months.”

Learning:

- “This module reinforces the skills needed for your target store tier.”

---

6. Micro-Personalisation Without Overreach

Tailor Shift must never feel intrusive or algorithmically manipulative.

Allowed:

- highlighting relevant roles
- adjusting order
- surfacing insights
- using maturity and context

Not allowed:

- predicting salaries
- inferring sensitive traits
- nudging toward specific brands
- making prescriptive claims (“you should apply”)
- using behavioural tracking
- dark patterns

Neutrality is mandatory.

---

7. Privacy Constraints (Strict)

7.1 Compensation alignment is always abstract

Only tags are used:

- “Within your range”
- “Above your current level”

Never:

- numbers
- ranges
- values

7.2 Assessment answers never surface

Only capability summary appears.

7.3 Brand view always filtered

Brand sees:

- experience blocks
- role level
- capability summary
- divisions
- geography
- store-tier exposure

Never:

- preferences
- compensation
- assessment answers
- learning history
- projections
- personal insights

7.4 No cross-Talent comparisons

No ranking of “best Talent”, only relevance.

---

8. Versioning & Testing

Version Format:

`PersonalisationEngine_v1_2026_Q1`

Testing:

- deterministic output testing
- 20 synthetic profiles (junior → senior)
- preference variation scenarios
- device context variation
- privacy boundary tests

CHAPTER 13 — INTERACTION ENGINE (V5 CANON)

Discrete, structured, consent-based introductions between Talents and Houses.

---

0. Purpose — Why an Interaction Engine Exists

Luxury recruitment demands:

- discretion,
- controlled visibility,
- mutual respect,
- privacy gates,
- no unwanted contact,
- clear consent,
- and fully auditable interaction states.

The Interaction Engine governs:

- how Brands express interest in Talents,
- how Talents express interest in Opportunities,
- how introductions are requested, reviewed, accepted, declined,
- how contact information is unlocked securely,
- how interaction histories remain private and in the Talent’s control.

It is the **connective tissue** between Talent and Brand.

---

1. Core Principles (Non-Negotiable)

The Interaction Engine is built on seven strict principles:

1.1 Mutual Consent

No introduction is valid unless **both sides** accept.

1.2 Privacy by Default

Before acceptance:

- Talent contact hidden
- Brand recruiter contact hidden
- sensitive profile data never accessible
- assessment answers never revealed
- compensation always masked

1.3 Talent Sovereignty

Talent always has the final decision.

1.4 No Cross-Party Leakage

No party can see any personal data about anyone else unless consent is explicit.

1.5 Deterministic State Machine

Interactions move through a controlled set of states.

1.6 Explainable & Auditable

All states and transitions are clear and recorded.

1.7 Luxury-Appropriate Etiquette

All messaging is clean, neutral, and respectful.

---

2. Entities — What the Engine Manages

The Interaction Engine manipulates one canonical entity:

Interaction {

- `id: UUID`
- `talent_id: UUID`
- `brand_id: UUID`
- `opportunity_id: UUID | null`
- `origin: 'talent' | 'brand'`
- `status: 'sent' | 'requested' | 'accepted' | 'declined'`
- `created_at, updated_at`
  }

Constraints:

- `(talent_id, opportunity_id)` must be unique (no duplicates)
- Only Talent or Brand admins can create interactions
- Only parties involved can update status

---

3. The Interaction Lifecycle (Canonical State Machine)

The Interaction Engine uses a **4-state deterministic lifecycle**:

```
requested → accepted → (contact unlock)
sent      → declined
```

Here are the valid transitions:

3.1 INITIATION

Case 1 — Talent expresses interest in an Opportunity

State:

```
requested
origin = 'talent'
status = 'requested'
```

Description:

- Talent clicks “Express interest discreetly”
- Interaction is created
- Brand sees the Talent profile in a **masked** state

Case 2 — Brand expresses interest in a Talent

State:

```
sent
origin = 'brand'
status = 'sent'
```

Description:

- Brand clicks “Request introduction”
- Talent receives a notification
- Brand sees Talent as “Pending review”

3.2 RESPONSE

Case 3 — Talent responds to Brand request

- Accept → `accepted`
- Decline → `declined`

Case 4 — Brand responds to Talent request

- Accept → `accepted`
- Decline → `declined`

3.3 MUTUAL ACCEPTANCE → CONTACT UNLOCK

State: `accepted`

When both sides accept:

- Talent sees:
  - brand recruiter contact
  - brand email
  - (optional) brand instructions for next steps
- Brand sees:
  - Talent email

    - Talent phone (optional)
    - Talent preferred contact method

Privacy Note:

Nothing beyond business-relevant fields is revealed.

UX Note:

Contact appears in a structured, elegant card.
No chat module, no messaging thread, no conversational UI.

3.4 DECLINED

State: `declined`

Once declined by either party:

- Interaction is frozen
- Neither side can reopen it
- No contact ever revealed

UX:

- clear, respectful message
- no guilt-tripping
- no pressure to reconsider

---

4. What Each Party Sees — Before & After Acceptance

Detailed visibility rules.

4.1 Talent View (Before Acceptance)

Talent sees:

- Brand name
- Opportunity details (if applicable)
- Public brand profile
- No recruiter details
- No internal notes
- No brand-level insights
- No candidate ranking

4.2 Brand View (Before Acceptance)

Brand sees:

- Talent name
- Talent role level
- divisions
- experience block summary
- store tier exposure
- assessment capability summary
- Talent geography
- No preferences
- No compensation
- No learning history

- No projections
- No contact details

4.3 After Acceptance (Both Sides)

Exposed:

- name
- email
- phone (if provided)
- preferred contact method

Never exposed:

- compensation
- assessment answers
- career preferences
- full learning history
- raw profile fields not intended for brand view

---

5. UX FLOW — Interaction Experience

5.1 Talent Initiated

1. Talent: “Express interest”
2. Brand sees “New Applicant”
3. Brand: “Review Profile” → “Accept Introduction”

4. Talent receives notification
5. Talent accepts or declines
6. Contact unlocks only if Talent accepts

5.2 Brand Initiated

1. Brand: “Request introduction”
2. Talent receives notification
3. Talent accepts or declines
4. If accepted:
   - Brand sees Talent contact
   - Talent sees Brand contact

5.3 Accept / Decline UI Requirements

Accept:

Button: **“Accept Introduction”**
Copy:

> “Your contact details will be shared. You remain in control.”

Decline:

Button: **“Decline”**
Copy:

> “The introduction will not proceed.”

No pressure.

No “Are you sure?” loops.

---

6. Interaction Engine Logic (Deterministic Rules)

Below is the exact pseudocode.

6.1 Creation (Talent-originated)

```
if no existing interaction for (talent, opportunity):
    create_interaction(
        origin='talent',
        status='requested',
        opportunity_id,
        brand_id
    )
else:
    return existing_interaction
```

6.2 Creation (Brand-originated)

```
if no existing interaction for (talent, opportunity):
    create_interaction(
        origin='brand',
        status='sent',
        opportunity_id,
        brand_id
    )
else:
    update status only if talent previously initiated
```

6.3 Acceptance

```
if (status == 'requested' and brand_accept) OR
   (status == 'sent' and talent_accept):
        status = 'accepted'
```

6.4 Decline

```
status = 'declined'
freeze interaction
```

---

7. Integration With Other Engines

7.1 Matching Engine

- Triggers interaction suggestions
- Provides relevance logic

7.2 Assessment Engine

- No direct exposure of answers
- Only high-level capability summary used

7.3 Learning Engine

- No direct exposure
- Interaction does not affect learning recommendations

7.4 Projection Engine

- Projection remains Talent-only
- Never exposed to Brands

7.5 Personalisation Engine

- Surfaces interaction states on dashboards
- Sorts pending introductions
- Highlights introductions requiring response

8. Fraud & Misuse Prevention

Luxury ecosystems require integrity.

Rules:

- No mass outreach: max X requests / day per Brand
- No re-request after decline
- No bulk interactions
- No visibility into rejection reasons
- No ranking of Talents beyond relevance

Audit trails stored with minimal, non-sensitive metadata:

- timestamp
- action
- actor

No behavioural analytics.
No ML-based pattern detection.

9. Versioning & Testing

Version Format:

`InteractionEngine_v1_2026_Q1`

Tests:

- full state machine transitions

- privacy boundary tests
- RBAC enforcement
- acceptance/decline flows
- UX snapshots

CHAPTER 14 — OPPORTUNITY TEMPLATES & ROLE DEFINITIONS (V5 CANON)

Universal, maison-agnostic role structures and opportunity templates for luxury retail.

---

0. Purpose — Why Opportunity Templates Exist

Before Brands post real opportunities, Tailor Shift uses **standardized, maison-agnostic templates** to:

- provide early insights to Talents,
- power the Matching Engine in V1,
- train Talents on how opportunity structure works,
- ensure the projection engine has consistent data,
- create a reliable UX for browsing roles even without Brand activity.

Later, in V2+, these templates coexist with real Brand-created roles.

This chapter defines:

1. The canonical **role definitions** (L1–L8)
2. The maison-agnostic **opportunity templates**

3. How templates are used in Matching, Assessment, Projection, and UX
4. How templates evolve into real Opportunities
5. The privacy & display conventions for templates vs real roles

---

1. Canonical Role Definitions (L1–L8)

(Aligned with MCS Role Ladder from Ch.1)

Every real-world job maps into one of these **eight normalized role levels**.

This ensures:

- consistent matching,
- predictable capability requirements,
- comparable scoring,
- transparent projection.

Each Role Definition includes:

- canonical responsibilities
- required behavioural signals
- expected capability maturity
- division variations
- store tier variations

L1 — Sales Advisor

Essence: Execute service rituals with consistency.

Typical Responsibilities

- greet customers, control floor pacing
- assist in product selection
- maintain presentation
- apply selling ceremony correctly
- support BOH when required

Capabilities

- strong service
- basic product knowledge
- emerging clienteling signals
- basic operations literacy

L2 — Senior Advisor

Essence: Elevate service and influence the floor subtly.

Responsibilities

- diagnose needs quickly
- adapt storytelling
- support product launches
- guide new FOH team members

Capabilities

- advanced service
- growing clienteling discipline
- accuracy in BOH routines

- priority management

L3 — Department / Floor Manager

Essence: Coordinate teams and maintain rhythm.

Responsibilities

- supervise advisors
- ensure service quality
- direct floor flow
- coordinate with BOH
- handle escalations

Capabilities

- leadership signals
- operations discipline
- coaching potential
- strong product fluency

L4 — Assistant Store Manager (ASM)

Essence: Operate the store at N-1 level.

Responsibilities

- support Store Manager
- lead shifts
- manage routines & compliance

- coach advisors
- ensure sales floor execution

Capabilities

- leadership maturity
- clienteling oversight
- operations governance
- team pacing

L5 — Store Manager

Essence: Own the store commercially and operationally.

Responsibilities

- operations ownership
- people development
- sales performance
- service rituals oversight
- coordination with HQ

Capabilities

- high leadership
- strong business acumen
- advanced operations
- clienteling governance

L6 — Flagship Director

Essence: Govern multi-division, high-complexity operations.

Responsibilities

- multi-level teams
- VIP systems
- cross-division coordination
- commercial pacing
- succession planning

Capabilities

- high capability maturity
- strategic operations
- people development
- VIC management

L7 — Cluster / Area Manager

Essence: Oversee multiple boutiques.

Responsibilities

- store performance governance
- cross-store talent pipeline
- alignment with strategy
- operational standardization
- coaching SMs

Capabilities

- multi-store leadership
- commercial strategy
- talent development
- geographic mobility

L8 — Retail / Regional Director

Essence: Govern market-level retail function.

Responsibilities

- strategic planning
- retail transformation
- coordination across regions
- performance leadership
- senior stakeholder alignment

Capabilities

- strategic leadership
- regional mobility
- advanced business intelligence
- cross-functional governance

---

2. Maison-Agnostic Opportunity Template Schema

Opportunity Templates use the exact **Opportunity** schema (Ch. 2) but fill required fields with **canonical retail expectations** instead of Brand-specific details.

All templates include:

- role level
- division(s)
- store type
- complexity tier
- geography (city + region)
- expected capabilities
- experience patterns
- compensation band (generic, non-specific)
- match guidelines
- store context
- “you may be a match if…” guidance

Templates appear in:

- Talent Dashboard
- `/talent/opportunities`
- Talent projections
- early Matching Engine tests
- product demos

---

3. Standard Templates (V5 — Global Canon)

We define **six universal templates** representing the most common luxury retail roles across global maisons.

Template 1 — Sales Advisor (L1)

Divisions:

- RTW, Leather Goods, Shoes, Beauty/Fragrance

Store Types:

- Main boutique, department corner, travel retail

Tier:

- T3–T5

Requirements:

- 0–2 years experience
- basic service literacy
- strong interest in division
- willingness to learn BOH routines

"You may be a match if…"

- your service and operations scores are solid
- you have at least one FOH experience block
- you've expressed interest in customer-facing roles

Template 2 — Senior Advisor (L2)

Divisions:

- RTW + LG
- Beauty / Watches (optional specialization)

Store Type:

- boutique + department store

Tier:

- T2–T4

Requirements:

- 2–3 years FOH experience
- storytelling ability
- emerging clienteling routines

Fit Indicators:

- strong service score
- growing clienteling score
- 1–2 FOH blocks

Template 3 — Department / Floor Manager (L3)

Divisions:

- multi-division retail (RTW+LG), Watches, Beauty

Store Type:

- multi-division boutiques

Tier:

- T2–T3

Requirements:

- early leadership exposure
- advanced FOH + basic Ops
- 3–5 years total experience

Fit:

- leadership signals
- previous shift coordination
- clienteling depth helpful

Template 4 — Assistant Store Manager (L4)

Divisions:

- multi-division retail
- shoes, LG, RTW, beauty

Store Type:

- boutiques, flagships

Tier:

- T2–T3

Requirements:

- shift leadership
- strong Ops + FOH

- coaching emerging
- 4–6 years experience

Fit:

- 1+ Leadership blocks
- operations excellence
- projection L3→L4

Template 5 — Store Manager (L5)

Divisions:

- RTW + LG
- Watches + HJ (specialist)
- Beauty Flagship

Store Type:

- T1–T2 boutiques
- complex stores

Requirements:

- full ops ownership
- people development
- multi-division expertise

Fit:

- strong leadership

- business acumen signals
- advanced Ops

Template 6 — Flagship Director (L6)

Divisions:

- RTW, LG, Shoes, Watches, HJ

Store Type:

- T1 flagship
- multi-division architecture

Requirements:

- multi-team leadership
- VIC governance
- complex operations
- strong brand ambassadorship

Fit:

- high capability scores
- strong projection (L5→L6)
- significant leadership blocks

---

4. Template Display Logic in UX

Templates appear in **Talent V1**:

4.1 In the Opportunity Feed

- Score shown (matching based on canonical expectations)
- Fit breakdown
- Division badges
- Store tier preview

4.2 In Opportunity Detail

- neutral description
- structured, elegant
- “You may be a match if…” section
- store archetype explanation
- no compensation numbers, only “category” (entry/mid/senior range)

4.3 After Real Opportunities Launch

- Templates remain as fallback
- but appear below real roles
- labelled “Maison-Agnostic Opportunity”

---

5. Role Definition → Opportunity Template Mapping

Each role definition must map cleanly to a template to ensure:

- deterministic Matching
- predictable UX behaviour

- stable projection logic

Example mapping:

Role Level	Template
L1	Sales Advisor Template
L2	Senior Advisor Template
L3	Floor Manager Template
L4	ASM Template
L5	Store Manager Template
L6	Flagship Director Template

Levels L7–L8 appear only in:

- projections
- learning paths
- brand-specific opportunities

But **no maison-agnostic templates** exist for L7–L8 to avoid complexity for V1.

---

6. Opportunity Template Lifecycle

Templates must follow:

6.1 Versioning

`OpportunityTemplate_v1_2026_Q1`

6.2 Maintenance Rules

- updated only during spec revisions

- aligned with MCS evolution
- division requirements refined annually
- tier expectations updated with new real-world data

6.3 Removal

Obsolete templates must be deprecated gracefully to preserve matching data stability.

---

7. Privacy & Compliance Rules

Templates involve **no personal data**.
Thus:

- safe to show to any Talent
- safe to use for matching demos
- do not expose store addresses or identifiable employer properties

Real opportunities follow stricter rules (Ch. 13).

CHAPTER 15 — TECH ARCHITECTURE OVERVIEW (V5 CANON)

A clean, composable, privacy-first architecture for a luxury-grade talent intelligence platform.

---

0. Purpose — What the Architecture Must Achieve

Tailor Shift's architecture must:

- guarantee **privacy-by-default**,

- support a **complex intelligence layer** (matching, assessment, learning, projection),
- provide **low-latency UI** across web and mobile,
- scale to thousands of Talents and House users,
- ensure **RBAC correctness**,
- maintain **zero-trust** data boundaries,
- integrate cleanly with **Auth.js**,
- allow deterministic reasoning engines,
- and support **future AI modules** safely.

The architecture is built around five pillars:

1. **Identity Layer** (Auth.js)
2. **Domain Database** (Tiger Cloud PostgreSQL)
3. **Access Control** (Row-Level Security + RBAC)
4. **Application Layer** (Next.js App Router)
5. **Intelligence Layer** (Edge Functions + server actions)

Everything is deterministic, auditable, and compliant.

---

1. High-Level Architecture Diagram

```
┌──────────────────────────────┐
 │          Web / Mobile          │
 │  Next.js App Router (SSR)  │
 └──────────────┬───────────────┘
                │
   Server Components / Actions
                │
 ┌──────────────▼───────────────┐
 │        Tailor Shift Core       │
 │      (Domain Logic Layer)      │
```

```
  └───────┬──────────────────────┘
          │
 ┌────────▼───────────────┐
 │   Intelligence Layer    │
 │ Matching / Assessment   │
 │ Learning / Projection   │
 └────────┬────────────────┘
          │
  ┌───────▼────────────────┐
  │ Tiger Cloud PostgreSQL  │
  │   + Row-Level Security  │
  │   + Canonical Entities  │
  └─────────┬───────────────┘
            │
        Auth.js
            │
  ┌─────────▼───────────────┐
  │   Identity & Sessions     │
  └──────────────────────────┘
```

This architecture enforces **single-responsibility** at every layer.

---

2. Identity Layer — Auth.js

2.1 Identity Model

Tailor Shift uses **Auth.js** to manage:

- user accounts,
- sessions,
- OAuth providers,
- password resets,
- email verification.

2.2 Session Handling

- session tokens stored securely
- no manual cookie manipulation
- no client-side auth logic
- SSR-safe session retrieval

2.3 Roles

Users have one of the canonical roles:

- `talent`
- `brand_admin`
- `recruiter`
- `admin`

Roles are assigned by:

- default (talent)
- creation of a Brand (brand_admin)
- admin tooling

2.4 RBAC

RBAC enforcement happens via:

- **middleware.ts**
- **server actions**
- **RLS policies**

Redundant enforcement ensures safety even if one layer misconfigures.

---

3. Domain Database — Tiger Cloud PostgreSQL

The database stores *all* product data.

3.1 Public Schema

Contains canonical entities:

- talents
- brands
- stores
- opportunities
- experience_blocks
- assessments
- learning_modules
- talent_learning_progress
- interactions

3.2 Canonical Types

Defined as Postgres enums:

- `division_enum`
- `region_enum`
- `store_type_enum`
- `complexity_tier_enum`
- `openness_enum`

3.3 JSONB for Flexible Fields

Used for:

- career preferences
- assessment summary
- opportunity requirements
- compensation profiles
- store context

3.4 Zero-Trust RLS

RLS enforces:

- Talent can only read/write own Talent row
- Brand Admin can only read/write brand they own
- Interaction visibility tightly restricted
- No unfiltered SELECTs
- All mutations guarded by role-based checks

---

4. Access Control — RLS + RBAC

4.1 Multi-Layer Enforcement

Security enforced at three levels:

1. **Auth.js** (identity + session)
2. **Next.js Middleware** (route gating)
3. **Postgres RLS** (data gating)

This forms a robust zero-trust boundary.

4.2 Example Policy — Talent Profile

```
SELECT * FROM talents
WHERE id = auth.uid()
   OR id IN (SELECT id FROM talent_public_view)
```

4.3 Example Policy — Opportunities

```
brand_id IN (
  SELECT id FROM brands
  WHERE owner_id = auth.uid()
)
```

---

5. Application Layer — Next.js App Router

(Full sitemap in Chapter 4)

5.1 Structure

- `app/(public)`
- `app/(talent)`
- `app/(brand)`
- `app/api/*`
- layouts for each segment
- server actions for mutations

5.2 Server Components First

All pages rendered on the server except:

- assessment wizard
- dynamic forms
- filters

- modals

5.3 Server Actions

Used for privileged operations:

- update Talent profile
- create Experience Block
- create Opportunity
- accept/decline interaction
- update preferences

5.4 No Monolithic Backend

Next.js = API + UI
Postgres = system of record
RLS = backend rules

No need for Express, NestJS, or separate servers.

---

6. Intelligence Layer — Deterministic Engines

From Ch. 7–12:

- Matching Engine
- Assessment Engine
- Learning Engine
- Projection Engine
- Personalisation Engine

6.1 Where Engines Run

Engines run in:

- server actions
- Edge Functions
- background jobs (future)

Never in the browser.

6.2 Engine I/O

Engines operate on:

- canonical entity objects
- typed vectors
- capability summaries
- opportunity signatures

Outputs flow into UI and storage as needed.

6.3 Versioning

Each engine declares:

```
"engine_version": "MatchingEngine_v1_2026_Q1"
```

This enables:

- regression testing
- rollout control
- explainability

---

7. Data Flow — End-to-End

7.1 Talent Signup

1. Auth.js creates account
2. DB trigger creates minimal Talent row
3. Redirect to onboarding
4. Talent fills identity, preferences
5. Profile completion triggers basic matching
6. Assessment populates capability summary
7. Projection & Learning engines update Dashboard

7.2 Brand Onboarding

1. Brand Admin signs up
2. Creates brand page
3. Adds stores
4. Posts opportunities
5. Matching Engine links talent universe

7.3 Matching Loop

1. Opportunity posted
2. Matching Engine runs → match snapshots
3. Brand Dashboard surfaces best matches
4. Talent Dashboard surfaces recommended roles
5. Personalisation Engine adjusts ordering
6. Interaction Engine governs introductions

---

8. DevOps Architecture

8.1 Environments

- local
- staging
- production

8.2 Deployments

- Vercel for Next.js
- Tiger Cloud for Postgres

8.3 CI/CD

PR checks:

- lint
- type-check
- test engines
- build
- RLS tests

8.4 Observability

- structured logs (server actions, errors)
- Sentry (client + server)
- DB slow query log
- interaction lifecycle logs

8.5 Secrets

- managed by Vercel
- never exposed in client
- service role keys used only in server-side contexts

---

9. Scalability Considerations

9.1 Performance Targets

- SSR pages < 150ms
- Engine calls < 250ms
- Interaction flows < 70ms
- DB queries < 20ms p50

9.2 Horizontal Scaling

Next.js on Vercel → auto scaled
Postgres → read replicas (future)

9.3 Query Optimization

- indexes on `brand_id`, `talent_id`
- JSONB path indexes for matching filters
- vectorized matching (future versions)

---

10. Privacy-by-Design Requirements

10.1 Compensation

NEVER exposed to Brands.

10.2 Assessment Answers

NEVER exposed to Brands or external APIs.

10.3 Talent Preferences

Visible only to Talent.

10.4 No third-party sharing

No external analytics with PII.

10.5 Data minimisation

Only canonical fields stored.
Raw assessment answers discarded after scoring.

---

11. Compliance (EU AI Act, GDPR)

11.1 Low-Risk Classification

All engines are:

- deterministic,
- rules-based,
- explainable,
- non-psychometric.

11.2 Transparency

Engines provide:

- score breakdowns
- rationale
- “no overclaiming” constraints

11.3 Data Rights

Talent may:

- export profile
- delete account
- anonymize data

Houses never receive sensitive fields.

CHAPTER 16 — IDENTITY & AUTHENTICATION (V5 CANON)

The identity contract of Tailor Shift. Secure, predictable, privacy-first, frictionless.

---

0. Purpose — What "Identity" Means in Tailor Shift

Identity is **the root of all data ownership** in Tailor Shift.
Every Talent, Brand Admin, Recruiter, and Admin is represented by a single **UserAccount**.

Authentication must be:

- frictionless
- globally consistent
- compliant
- safe
- structured
- non-leaking
- deterministic

Nothing in the product can bypass the identity layer.

---

1. Identity Layer — Auth.js (Canonical Source of Truth)

Tailor Shift uses **Auth.js** as the single identity provider.

1.1 What Auth.js manages

- account creation
- login / logout
- OAuth flows
- password resets
- email verification
- session creation
- secure cookies

1.2 Identity Object

Upon sign-up, a canonical `UserAccount` object exists with:

```
UserAccount {
  id: UUID
  email: string
  role: 'talent' | 'brand_admin' | 'recruiter' | 'admin'
  created_at: datetime
  updated_at: datetime
}
```

This object **cannot** be overridden by downstream logic.

---

2. Role Model — The Four Canonical Roles

Tailor Shift uses a **minimal, luxury-appropriate** RBAC model:

2.1 Talent (default)

Created automatically for every new user unless they explicitly sign up as a House.

2.2 Brand Admin

Assigned when a user:

- creates a Brand (Ch. 3.2)
  or
- is invited into a Brand admin team (future)

2.3 Recruiter

Brand role with limited permissions (future V2+).

2.4 Admin

Internal Tailor Shift role (never exposed externally).

Forbidden:

No “superuser” or cross-brand visibility.

---

3. Authentication Methods

3.1 Email + Password (Primary)

- frictionless mobile experience
- minimal fields
- instant signup

3.2 OAuth Providers (Optional)

Supported:

- Google

- LinkedIn
- Apple (V2)

OAuth redirects must resolve through:

`/auth/callback`

3.3 Password Reset

- Request via `/auth/reset-password`
- Link issued via Auth.js
- Reset finalisation handled server-side
- Silent session renewal after success

3.4 Email Confirmation

- required for full account activation
- auto-resent upon login attempt if pending

---

4. Session Management — SSR-First Design

Sessions are resolved on the server for **every protected route**, ensuring:

- stable SSR
- correct RBAC
- no client-side hacks
- no duplicated identity state

4.1 Session Retrieval

Pages use:

```
import { auth } from "@/lib/auth";
const session = await auth();
```

4.2 Session Guarantees

- session includes role
- session includes `user_id`
- session is cryptographically signed
- session refreshed automatically
- no client-side JWT decoding

---

5. RBAC — Route Protection & Access Boundaries

Enforced in **three layers**:

Layer 1 — Route Gating (middleware.ts)

```
if url.startsWith("/talent") and role !== "talent":
   redirect("/login")

if url.startsWith("/brand") and role not in ["brand_admin",
"recruiter"]:
   redirect("/login")
```

This ensures **UX-level safety**.

Layer 2 — Server Actions

Every mutation is wrapped in a type-safe server action:

```
if (session.role !== "talent") throw "Forbidden";
```

Layer 3 — Tiger Cloud Row-Level Security (RLS)

The final, unbreakable boundary.

Example: Talent profile:

```
CREATE POLICY talent_is_self ON public.talents
FOR SELECT USING (auth.uid() = id);
```

Brand profile:

```
CREATE POLICY brand_owner_only ON public.brands
FOR UPDATE USING (auth.uid() = owner_id);
```

RLS is the ultimate guarantee that no frontend mistake leaks data.

---

6. The Identity Lifecycle — Step by Step

6.1 Talent Signup Flow

1. User signs up
2. `UserAccount` created (role=talent)
3. `Talent` entity auto-created
4. Redirect to onboarding
5. Sessions created automatically
6. All pages under `/talent/*` now accessible

6.2 Brand Admin Signup Flow

1. User chooses “House / Employer”
2. Signs up
3. `UserAccount(role=brand_admin)`
4. Creates Brand profile
5. Adds stores / opportunities
6. Gains full access to `/brand/*`

6.3 Logout Flow

- session cleared
- cookies invalidated
- redirect to homepage

---

7. Identity & Data Ownership Rules

7.1 Talent Owns:

- all Talent profile fields
- experience blocks
- preferences
- compensation profile
- assessment results
- learning history
- projections

7.2 Brand Owns:

- brand page
- store network
- opportunities
- brand users

7.3 Shared Visibility Only After Consent

Contact unlocks only when the Interaction Engine transitions to `accepted`.

7.4 Never Shared:

- Talent compensation
- Talent assessment answers
- Talent preferences
- Talent projections
- Talent learning progress

---

8. Security Guarantees

8.1 Zero-Trust Model

Every domain table protected with RLS.

8.2 No Client-Side Secrets

Auth tokens stored in HttpOnly cookies, managed by Auth.js.

8.3 No “Super Admin” Backdoor

Admin role can only access system maintenance features; no user data can be read without proper RLS.

8.4 OAuth Safety

Callback domain locked to production/staging URLs.

8.5 Password Storage

Managed by Auth.js → Argon2 / bcrypt depending on provider.

8.6 Session Hijacking Prevention

- rotating tokens
- strict same-site cookies
- no localStorage token usage

---

9. Compliance (GDPR + EU AI Act)

9.1 GDPR

- Data minimisation (only canonical fields stored)
- Right to deletion → Talent anonymised, account removed
- Right to export → machine-readable JSON
- No personal data used for learning or model training
- No behavioural profiling

9.2 EU AI Act

Identity & Authentication does *not* fall under high-risk classification:

- fully deterministic
- no inference
- no sensitive attribute processing
- used only for access control

Fully compliant.

10. Integration With Other Systems

10.1 With Matching Engine

Uses role-level, divisions, and preferences but no personal identifiers.

10.2 With Interaction Engine

Identity determines which party may accept or decline introductions.

10.3 With Assessment Engine

Assessment linked by Talent ID; answers never exposed.

10.4 With Learning Engine

Eligibility and access controlled per Talent.

10.5 With Projection

Projection uses canonical Talent profile, not identity fields.

---

11. Testing & Hardening

11.1 Unit Tests

- role gating
- session validity
- privilege escalation attempts
- expired sessions

11.2 Integration Tests

- RLS tests for all access points
- server action permission tests

11.3 Penetration Tests

- brute-force
- token replay
- CSRF
- OAuth misconfiguration

CHAPTER 17 — DATABASE SCHEMA (POSTGRESQL + RLS, V5 CANON)

The authoritative data model for Tailor Shift. Canonical, secure, explainable, future-proof.

---

0. Purpose — Why a Canonical Schema Exists

Luxury retail intelligence requires:

- structured roles (L1–L8),
- structured stores (T1–T5),
- structured divisions,
- structured experience blocks,
- structured preferences,
- strict privacy,
- deterministic logic,
- secure access boundaries (zero-trust).

This schema is:

- the single source of truth

- fully normalised,
- built for performance,
- governed by strict RLS
- aligned with the intelligence layer
- the foundation for all engines

Nothing in the app may store data *outside* this schema.

---

1. PostgreSQL Environment

- Database: **Tiger Cloud PostgreSQL 15+**
- Schema: **public** (domain data), **auth** (Auth.js)
- Extensions used:
  - `pgcrypto`
  - `uuid-ossp`
  - `citext` (emails)
  - `pg_trgm` (search optimization)
  - `btree_gin` (JSONB indexing)
  - ***No vector extensions required for V5***

---

2. Canonical ENUM Types

ENUMs enforce the luxury retail ontology.

```
CREATE TYPE division_enum AS ENUM (
  'fashion', 'leather_goods', 'shoes', 'beauty',
  'fragrance', 'watches', 'high_jewelry', 'eyewear', 'accessories'
);

CREATE TYPE region_enum AS ENUM
('emea','americas','apac','middle_east');

CREATE TYPE store_type_enum AS ENUM (
  'flagship','main_boutique','department_corner',
  'travel_retail','outlet','pop_up','franchise','partner','hJ_salon'
);

CREATE TYPE complexity_tier_enum AS ENUM ('T1','T2','T3','T4','T5');

CREATE TYPE openness_enum AS ENUM (
  'actively_looking','open_to_opportunities','not_looking'
);
```

---

3. TABLE: talents

The core entity for all Talent-related data.

```
CREATE TABLE public.talents (
  id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,

  first_name        TEXT NOT NULL,
  last_name         TEXT NOT NULL,
  headline          TEXT,
  bio               TEXT,

  location_region   region_enum,
  location_country  TEXT,
  location_city     TEXT,

  languages         JSONB DEFAULT '[]',      -- [{code, level}]

  years_experience  INT,
  current_role_level INT,                     -- L1–L8
  official_title    TEXT,
  divisions_experience division_enum[],

  openness_status   openness_enum DEFAULT 'open_to_opportunities',
  openness_comment  TEXT,

  career_preferences JSONB,                   -- unified preference
object

  assessment_summary JSONB,                   -- capability summary
  badges             JSONB DEFAULT '[]',

  compensation_profile JSONB,                 -- confidential

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Notes

- Talent → UserAccount: **1:1**
- Preferences stored as JSONB (flexible but structured)
- Compensation stored encrypted-at-rest, never exposed

---

4. TABLE: experience_blocks

```
CREATE TABLE public.experience_blocks (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  talent_id UUID REFERENCES public.talents(id) ON DELETE CASCADE,

  block_type TEXT NOT NULL,  -- FOH / BOH / Leadership / Clienteling
/ Ops / Business
  description TEXT,
  achievements TEXT[],
  years_in_block INT,
  divisions_handled division_enum[],
  complexity_tier complexity_tier_enum,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Notes

- Critical for Matching, Assessment, Projection
- Multiple blocks per Talent

---

5. TABLE: brands

```
CREATE TABLE public.brands (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  owner_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,

  name TEXT NOT NULL,
  group_type TEXT,
  is_independent BOOLEAN DEFAULT FALSE,

  divisions division_enum[],
  hq_region region_enum,
  hq_country TEXT,
  hq_city TEXT,

  employer_page JSONB,      -- description, visuals, etc.

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

Notes

- Brand Admin ≠ Talent
- Brand owns stores and opportunities

---

6. TABLE: stores

```
CREATE TABLE public.stores (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  brand_id UUID REFERENCES public.brands(id) ON DELETE CASCADE,

  name TEXT NOT NULL,
  region region_enum,
  country TEXT NOT NULL,
  city TEXT NOT NULL,

  store_type store_type_enum,
  size_class TEXT,
  complexity_tier complexity_tier_enum,
  divisions_present division_enum[],

  ownership TEXT,            -- direct / franchise / partner
  footfall_profile TEXT,     -- high/medium/low

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

---

7. TABLE: opportunities

```
CREATE TABLE public.opportunities (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  brand_id UUID REFERENCES public.brands(id) ON DELETE CASCADE,
  store_id UUID REFERENCES public.stores(id),

  title TEXT NOT NULL,
  normalized_role_level INT NOT NULL,
  divisions_required division_enum[],

  requirements JSONB,            -- skills, languages, years, etc.
  compensation JSONB,            -- *band only, Talent never sees raw
Talent comp*
  status TEXT DEFAULT 'open',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

---

8. TABLE: assessments

Stores individual assessment runs.

```
CREATE TABLE public.assessments (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  talent_id UUID REFERENCES public.talents(id) ON DELETE CASCADE,

  version TEXT,
  answers JSONB,       -- NEVER exposed to Houses
  scores JSONB,        -- {service, clienteling, operations,
leadership}
  insights JSONB,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

---

9. TABLE: learning_modules

```
CREATE TABLE public.learning_modules (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),

  title TEXT NOT NULL,
  category TEXT NOT NULL,            -- service / clienteling /
operations / leadership / business
  duration_minutes INT,
  summary TEXT,
  content_type TEXT,                 -- video / text / slides
  content_ref TEXT,                  -- URL or static reference
  quiz JSONB,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

10. TABLE: talent_learning_progress

```
CREATE TABLE public.talent_learning_progress (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  talent_id UUID REFERENCES public.talents(id) ON DELETE CASCADE,
  module_id UUID REFERENCES public.learning_modules(id) ON DELETE
CASCADE,

  completed_at TIMESTAMP WITH TIME ZONE
);
```

11. TABLE: interactions

Interaction Engine (Ch. 13) state machine

```
CREATE TABLE public.interactions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),

  talent_id UUID REFERENCES public.talents(id) ON DELETE CASCADE,
  brand_id UUID REFERENCES public.brands(id) ON DELETE CASCADE,
  opportunity_id UUID REFERENCES public.opportunities(id),

  origin TEXT,                -- 'talent' | 'brand'
  status TEXT,                -- 'requested' | 'sent' | 'accepted' |
'declined'

  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

  UNIQUE (talent_id, opportunity_id)
);
```

12. VIEWS — Controlled Exposure

12.1 talent_public_view

Exposes only fields safe for Brand viewing:

```
CREATE VIEW public.talent_public_view AS
SELECT
  id,
  first_name,
  last_name,
  current_role_level,
  divisions_experience,
  years_experience,
  location_region,
  location_country,
  location_city,
  assessment_summary,
  openness_status
FROM public.talents;
```

Hidden:

- preferences
- compensation
- assessment answers
- learning progress
- projections
- contact details

Talent controls visibility via Interaction Engine.

---

13. RLS — Zero Trust Security Model

13.1 Enable RLS

```
ALTER TABLE public.talents ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.brands ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.stores ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.opportunities ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.experience_blocks ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.assessments ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.interactions ENABLE ROW LEVEL SECURITY;
```

13.2 TALENTS — RLS Policies

```
-- Talent can read their own row
CREATE POLICY talents_self_select ON public.talents
FOR SELECT USING (auth.uid() = id);

-- Public profile visible to brands but ONLY via the controlled view
CREATE POLICY talents_public_view ON public.talents
FOR SELECT USING (false);  -- base table never readable directly
```

13.3 BRANDS — RLS Policies

```
CREATE POLICY brands_owner_select ON public.brands
FOR SELECT USING (owner_id = auth.uid());

CREATE POLICY brands_owner_update ON public.brands
FOR UPDATE USING (owner_id = auth.uid());

CREATE POLICY brands_insert ON public.brands
FOR INSERT WITH CHECK (owner_id = auth.uid());
```

13.4 OPPORTUNITIES — RLS Policies

```
CREATE POLICY opp_brand_owner_select ON public.opportunities
FOR SELECT USING (
  brand_id IN (SELECT id FROM public.brands WHERE owner_id =
auth.uid())
);

CREATE POLICY opp_brand_owner_mutation ON public.opportunities
```

```
FOR INSERT, UPDATE, DELETE USING (
  brand_id IN (SELECT id FROM public.brands WHERE owner_id =
auth.uid())
);
```

13.5 INTERACTIONS — RLS Policies

```
-- Talent can see their own interactions
CREATE POLICY interactions_talent_view ON public.interactions
FOR SELECT USING (talent_id = auth.uid());

-- Brand can see interactions linked to their brand
CREATE POLICY interactions_brand_view ON public.interactions
FOR SELECT USING (
  brand_id IN (SELECT id FROM public.brands WHERE owner_id =
auth.uid())
);

-- Mutations allowed only by involved parties
CREATE POLICY interactions_mutation ON public.interactions
FOR UPDATE USING (
  talent_id = auth.uid()
  OR brand_id IN (SELECT id FROM public.brands WHERE owner_id =
auth.uid())
);
```

---

14. Indexing Strategy (Performance-Tuned)

Primary indexes:

- `talents(id)`
- `brands(owner_id)`
- `stores(brand_id)`

- `opportunities(brand_id)`
- `experience_blocks(talent_id)`
- `assessments(talent_id)`

Secondary indexes:

```
CREATE INDEX idx_opportunities_role_level ON
opportunities(normalized_role_level);
CREATE INDEX idx_opportunities_divisions ON opportunities USING GIN
(divisions_required);
CREATE INDEX idx_talents_divisions ON talents USING GIN
(divisions_experience);
CREATE INDEX idx_experience_blocks_divisions ON experience_blocks
USING GIN (divisions_handled);
CREATE INDEX idx_talent_openness ON talents(openness_status);
```

JSONB indexes:

```
CREATE INDEX idx_talent_preferences_jsonb ON talents USING GIN
(career_preferences);
CREATE INDEX idx_opportunity_requirements_jsonb ON opportunities
USING GIN (requirements);
```

---

15. Migrations & Versioning

Migration Rules:

- one migration per schema change
- SQL files stored in `/migrations`
- versioned with timestamps
- no web-dashboard manual edits unless mirrored

Version Format:

CHAPTER 18 — API SURFACE & SERVER ACTIONS (V5 CANON)

A minimal, secure, deterministic interface for all Tailor Shift operations.

---

0. Purpose — Why This Chapter Exists

Tailor Shift is deliberately **not** a REST-heavy or RPC-heavy system.

- The **Next.js App Router** acts as the orchestration layer.
- **Server Actions** handle server-side mutations.
- **RLS** protects the database.
- **Server Components** fetch data securely.
- **API Routes** only exist for *integration* and *webhooks*.
- **Edge Functions** exist only for heavy compute or batch processes.

The goal is:

- zero leakage of sensitive data
- zero ad-hoc API surfaces
- no untyped endpoints
- no implicit database shortcuts
- minimal attack surface
- deterministic flows

This chapter defines **every permitted interaction pattern** in Tailor Shift.

---

1. Architecture Summary

Tailor Shift exposes functionality in **four distinct layers**:

1. **Server Components**
   - default for all page rendering
   - SSR data fetching
   - session-aware UI
2. **Server Actions** *(primary mutation interface)*
   - type-safe async functions
   - run server-side only
   - respect RBAC & RLS
   - used for forms, updates, interaction changes
3. **API Routes** *(rare)*
   - used for webhooks
   - opportunity batch recomputation
   - internal automation triggers
4. **Edge Functions** *(future optional)*
   - for batch matching
   - for analytics rollups
   - for heavy data tasks

Everything else is forbidden.

---

2. Canonical Data Access Pattern

All data operations happen via:

- **PRISMA (optional)** + direct SQL,
  OR
- **Postgres client (preferred)**,
  AND
- **RLS** ensuring domain boundaries.

Client-side code **never** touches the DB directly.

---

3. Allowed Interaction Patterns

3.1 GET — Server Components

Used for **all data fetches** that do not mutate state.

Examples:

- getTalentProfile()
- listOpportunities()
- getBrandStores()
- getOpportunity(id)
- getLearningModules()
- getAssessmentSummary()
- getInteractionStatus()

All of these are implemented as:

```
// server component
const session = await auth();
const data = await db.query(...); // safe behind RLS
```

No fetch to `/api/*` from UI.
No client-side fetching of privileged data.

3.2 POST/PUT/PATCH — Server Actions

Server Actions are the **primary mutation mechanism**.

They are:

- typed
- protected by RBAC
- executed server-side
- free of client secrets
- compatible with forms
- simple, modern, minimal

Examples:

- `updateTalentProfileAction`
- `updateTalentPreferencesAction`
- `addExperienceBlockAction`
- `completeAssessmentAction`
- `createBrandAction`
- `createStoreAction`
- `createOpportunityAction`
- `updateOpportunityAction`

- `requestIntroductionAction`
- `acceptIntroductionAction`
- `declineIntroductionAction`
- `completeLearningModuleAction`

Each action:

- validates the session
- checks role
- executes safe DB mutations
- handles RLS automatically
- returns typed output

This is the **canonical Tailor Shift API**.

---

4. Detailed Server Action Specification

4.1 Talent Server Actions

updateTalentProfileAction

```
input: { first_name, last_name, headline, bio, languages }
auth: talent only
effect: UPDATE talents SET ...
```

updateTalentPreferencesAction

```
input: career_preferences JSON
auth: talent only
effect: UPDATE talents.career_preferences
```

updateOpennessAction

```
input: openness_status, openness_comment
```

```
auth: talent only
```

addExperienceBlockAction

```
input: block_type, description, achievements[], years, divisions,
tier
auth: talent only
```

removeExperienceBlockAction

```
input: block_id
auth: talent only
```

completeAssessmentAction

```
input: answers[]
auth: talent only
effect:
  - compute capability summary
  - INSERT INTO assessments
  - UPDATE talents.assessment_summary
```

4.2 Brand Server Actions

createBrandAction

```
input: name, group_type, divisions, hq_region, hq_country, hq_city
auth: brand_admin
effect: INSERT INTO brands(owner_id=session.user_id)
```

updateBrandProfileAction

```
input: divisions, employer_page
auth: brand_admin
```

createStoreAction

```
input: brand_id, name, type, tier, divisions_present, geography
auth: brand_admin
effect: INSERT INTO stores
```

createOpportunityAction

```
input: brand_id, store_id, role_level, divisions_required,
       requirements, compensation, status
auth: brand_admin
effect: INSERT INTO opportunities
```

updateOpportunityAction

```
auth: brand_admin
effect: PATCH opportunity
```

4.3 Interaction Server Actions (Talent ↔ Brand)

requestIntroductionAction

(Brand → Talent)

```
input: talent_id, opportunity_id
auth: brand_admin
effect:
  create or update interaction(status='sent')
```

expressInterestAction

(Talent → Brand)

```
input: opportunity_id
auth: talent
effect:
  create or update interaction(status='requested')
```

acceptIntroductionAction

```
auth: talent OR brand_admin
effect: if counterpart already expressed interest →
status='accepted'
```

declineIntroductionAction

```
auth: talent OR brand_admin
effect: status='declined'
```

4.4 Learning Server Actions

markLearningModuleCompleteAction

```
input: module_id
auth: talent only
effect:
  INSERT INTO talent_learning_progress
```

---

5. API Routes (Minimal Surface)

API routes only exist for:

5.1 Webhooks

- email events (password reset)
- OAuth token renewal (provider events)

5.2 Batch Matching Trigger

```
POST /api/matching/recompute
auth: admin only
```

5.3 Health Check

```
GET /api/health
```

5.4 External Integrations (future)

- analytics ingestion
- inbound integration from ATS systems

Forbidden:

- CRUD APIs for Talent/Brand/Opportunities
- client-side access to any sensitive resource

---

6. Edge Functions (Optional)

To be used **only** for:

- nightly recomputation of match snapshots
- analytics rollups
- global trending metrics
- background computation for projection scaling

Never used for:

- authentication
- direct Talent/Brand interactions
- client-facing mutations

---

7. Error Handling — Canonical Format

All server actions MUST return errors in the following format:

```
{
  ok: false,
  error: {
    code: 'AUTH_ERROR' | 'FORBIDDEN' | 'VALIDATION_ERROR' |
'RLS_DENIED' | 'UNKNOWN',
    message: string
  }
}
```

UX surfaces errors in calm, neutral language:

- “Action not permitted.”
- “We couldn’t save your changes.”

- “You do not have access.”

---

8. Logging & Observability

8.1 Server Action Logs

- anonymized
- no personal data
- entry: action name, duration, error state

8.2 DB Query Logs

- slow query tracking
- RLS denials

8.3 Interaction Logs

Minimal metadata only:

- interaction_id
- action timestamp
- actor (talent/brand)

---

9. Performance Budget

Server Action Latency Targets

- simple updates: < 120 ms
- assessment scoring: < 200 ms

- matching function: < 250 ms
- interaction transitions: < 70 ms

API Limits

- max 20 interactions/day per brand
- max 10 introduction requests from a Talent/day

---

10. Prohibited API Patterns

The following are explicitly banned:

- REST endpoints exposing Talent data
- GraphQL layer
- client-side write operations
- serverless functions returning sensitive objects
- database access via client
- uncontrolled streaming
- raw SQL in client code
- exposing compensation, projections, or assessment answers publicly

CHAPTER 19 — FRONTEND ARCHITECTURE (V5 CANON)

Server-first, luxury-grade, deterministic frontend architecture using the Next.js App Router.

---

0. Purpose — Why This Architecture Exists

The frontend must:

- express the **luxury-grade UX system**,
- remain **server-driven**,
- enforce **privacy and RBAC**,
- allow **intelligence engines** to surface insights elegantly,
- avoid frontend complexity and fragile state,
- feel **calm, fast, structured**,
- match the **tone and precision** of luxury retail.

The architecture is:

- predictable,
- fully typed,
- aligned with data ownership,
- aligned with zero-trust backend boundaries,
- built for long-term maintainability.

---

1. High-Level Principles

1.1 Server Components by Default

All pages (`page.tsx`) are server components:

- SSR guarantees consistent state
- ensures correct session & RLS data

- prevents leaking sensitive fields
- boosts performance
- reduces client bundle size

1.2 Client Components Only When Necessary

Used only for:

- forms
- wizards (Assessment)
- filters
- micro-interactions (tabs, dropdowns)
- modals
- responsive components requiring browser APIs

1.3 Server Actions for Mutations

- no REST calls for internal logic
- no client-to-DB direct operations
- Actions handle validation + RBAC
- UI submits forms that call server actions

1.4 Zero Global State

There is **no Redux**, **no Zustand**, **no global context** for app data.

Everything should be SSR-rendered or fetched in server components.

Exception: micro state inside client components (tabs, input state).

1.5 Layout-Driven Architecture

Segment-level layouts:

- `(public)`
- `(talent)`
- `(brand)`

These define headers, sidebars, containers.

They ensure:
consistent structure → no UI drift.

1.6 Component-First Design

All UI must use the canonical components from the design system (Ch. 5):

- Card
- Input
- Button
- Badge
- Tabs
- ScoreBar
- MatchScore
- ProfileCompletionBar
- OpportunityCard
- TalentCard
- StoreCard

No custom per-page components unless approved.

---

2. Directory Structure — Canonical App Router Layout

The `/app` directory is structured exactly as defined in Chapter 4:

```
/app
├── layout.tsx
├── (public)
│   ├── layout.tsx
│   ├── page.tsx
│   ├── professionals/page.tsx
│   ├── brands/page.tsx
│   ├── about/page.tsx
│   ├── resources/page.tsx
│   ├── login/page.tsx
│   ├── signup/page.tsx
│   ├── terms/page.tsx
│   ├── privacy/page.tsx
│   └── ...
│
├── (talent)
│   ├── layout.tsx
│   ├── dashboard/page.tsx
│   ├── profile/**
│   ├── assessment/**
│   ├── learning/**
│   ├── opportunities/**
│   ├── community/page.tsx
│   └── settings/page.tsx
│
├── (brand)
│   ├── layout.tsx
│   ├── dashboard/page.tsx
│   ├── profile/**
│   ├── stores/**
│   ├── opportunities/**
│   ├── search/page.tsx
│   ├── talent/[id]/page.tsx
│   └── settings/page.tsx
│
└── api/**
```

No additional routes allowed.

---

3. Data Fetching Architecture (SSR-First)

3.1 How Pages Fetch Data

All page-level fetching uses:

```
const session = await auth();
const data = await db.query(...); // protected by RLS
```

Never:

- `fetch('/api/...')` from server components
- fetching via client-side queries for privileged data

3.2 Benefits

- correct RBAC
- correct data for SSR
- no race conditions
- minimal hydration mismatches

---

4. Server Actions — Frontend Integration

Forms submit directly to server actions:

```
<form action={updateTalentProfileAction}>
  ...
</form>
```

Or within client components:

```
'use client'

const formAction = useFormState(updateTalentPreferencesAction,
initialState);
```

4.1 Action Patterns

- optimistic UI only for non-sensitive fields
- errors handled through returned result objects
- never swallow RLS errors — expose clean user-friendly messages

4.2 Input Validation

Actions validate:

- types
- required fields
- business constraints (e.g., role levels must be L1–L8)

---

5. Component Architecture

Tailor Shift UI is structured into 3 layers of components.

5.1 Layer 1 — Foundations (Design System Primitives)

Located in `components/ui/`

Examples:

- `Button`
- `Input`
- `Textarea`
- `Select`

- `Card`
- `Badge`
- `Tabs`
- `Modal`
- `ScoreBar`
- `MatchScore`
- `ProgressRing`
- `Skeleton`

These use:

- Tailwind tokens
- minimal styling
- consistent spacing
- fully accessible markup

5.2 Layer 2 — Domain Components

Located in `components/domain/`

Examples:

- `TalentCard`
- `OpportunityCard`
- `StoreCard`
- `AssessmentRadarChart`

- `CapabilitySummary`
- `CareerProjectionCard`
- `LearningModuleCard`
- `InteractionStatusCard`

These encode **business meaning**, not raw UI.

5.3 Layer 3 — Flow Components (Client-Only)

Located in `components/flows/`

Examples:

- Assessment Wizard
- Multi-step onboarding
- Experience Block Creation form
- Opportunity creation wizard
- Store creation wizard
- Interaction acceptance modal

These exist only when necessary for user input.

---

6. State Management

6.1 Zero Global State

No Redux.
No Zustand.
No MobX.
No Apollo Client.

Global state in luxury-grade apps creates:

- hydration errors
- data drift
- privacy leaks
- inconsistent SSR

6.2 Allowed State

Only *local*, *ephemeral* state in client components:

```
const [view, setView] = useState("overview")
```

6.3 Server-State Authority

Server state is canonical.
Client state exists only for UX transitions.

---

7. Navigation Architecture

7.1 Public Segment

Header with elegant nav:

- Home
- Professionals
- For Houses
- About
- Login / Signup

7.2 Talent Segment

Sidebar (desktop) or top-bar (mobile):

- Dashboard
- Profile
- Assessment
- Learning
- Opportunities
- Settings

7.3 Brand Segment

- Dashboard
- Stores
- Opportunities
- Search Talent
- Settings

Navigation must be:

- calm
- predictable
- non-distracting
- left-aligned
- no icons unless necessary

---

8. Rendering & Performance Constraints

8.1 p50 Targets

- SSR pages: <150 ms
- opportunity listings: <180 ms
- Talent Dashboard: <200 ms
- Brand Dashboard: <200 ms

8.2 Code Splitting

Automatic via the App Router.

8.3 Hydration

Client components kept minimal to reduce JS payload.

8.4 Suspense + Skeleton UI

Used for:

- cards
- charts
- detail pages

Skeletons follow a strict minimalist aesthetic (light grey shapes).

---

9. Error Handling & Empty States

9.1 Error States

Neutral, calm:

- “Something went wrong.”
- “You do not have access.”
- “We couldn’t load this content.”

9.2 Empty States

- “No experience blocks added yet.”
- “No opportunities match your preferences.”
- “No stores have been created.”

9.3 Loading States

- Soft shimmer skeleton
- No animations beyond opacity

---

10. Accessibility Requirements

Every component must:

- meet WCAG 2.1 AA
- support keyboard navigation
- include semantic HTML
- ensure focus rings are visible
- avoid motion-heavy transitions

---

11. Internationalisation (i18n) Roadmap

V5:

- English-only spec
- consistent language tokens
- no dynamic translations

V6 (future):

- i18n infrastructure added
- translation files only for labels and static text
- engines remain language-agnostic

---

12. Security Requirements (Frontend)

Forbidden:

- storing tokens in localStorage
- client-side access to protected endpoints
- rendering sensitive fields in Client Components
- exposing session payload in console

Allowed:

- HttpOnly secure cookies (Auth.js)
- controlled hydration
- ephemeral form state

CHAPTER 20 — ERROR HANDLING, LOGGING & OBSERVABILITY (V5 CANON)

A calm, predictable, privacy-safe, fully observable architecture for operational excellence.

---

0. Purpose — Why This Chapter Exists

Tailor Shift’s audience—luxury houses, high-end retailers, and top professionals—expects:

- **stability**,
- **reliability**,
- **trust**,
- **discretion**,
- **predictability**,
- and **zero drama**.

Therefore:

- errors must be handled **gracefully**,
- sensitive information must **never** leak,
- logs must be **structured** and **non-PII**,
- monitoring must be **accurate**,
- observability must be **minimalist** but **complete**,
- failures must be explained with **neutral, composed language**.

This chapter establishes the **canonical system for Tailor Shift’s operational resilience**.

---

1. Global Error Philosophy

The platform uses a **Calm Error Doctrine**:

1. **Silent on sensitive details**
2. **Clear on what happened**
3. **Respectful to the user**
4. **Actionable for engineers**

5. **Never alarming or dramatic**

6. **Always privacy-preserving**

Users see *clean*, *simple*, *neutral* text.
Engineers see *structured logs*.

---

2. Error Surfaces Across the Application

Errors can appear in 4 layers:

1. **Frontend Rendering Layer (App Router SSR/CSR)**
2. **Server Action Layer**
3. **Database / RLS Layer**
4. **Intelligence Engines**

Each layer has strict patterns.

---

3. Error Types (Canonical Taxonomy)

All errors fall into one of seven categories:

3.1 AUTH_ERROR

- incorrect password
- invalid session
- expired token
- OAuth failure

User-facing:

> "Authentication failed. Please try again."

3.2 FORBIDDEN

Triggered by:

- RBAC violations
- RLS denials
- trying to access another entity’s data

User-facing:

> “You do not have access to this resource.”

Never show:

- table names
- SQL details
- internal references

3.3 VALIDATION_ERROR

Triggered by:

- missing form fields
- invalid role level
- invalid division entry
- malformed JSON

User-facing:

> “Some information is missing or incorrect.”

3.4 INPUT_FORMAT_ERROR

Triggered by:

- incorrect type
- malformed data
- non-JSON payload

User-facing:

> "The data format is invalid."

3.5 ENGINE_ERROR

Triggered by:

- matching engine failure
- assessment scoring failure
- projection misalignment
- internal mathematical inconsistency

User-facing:

> "We couldn't process this request. Please try again."

Logged with engine name + version.

3.6 DATABASE_ERROR

Triggered by:

- constraint violation
- transaction failure
- unexpected RLS rejection
- write conflict

User-facing:

> “The operation could not be completed.”

3.7 UNKNOWN_ERROR

Fallback for uncaught exceptions.

User-facing:

> “Something went wrong.”

---

4. Error Handling Patterns

4.1 Frontend (App Router)

Server Component Errors

Handled via:

- `error.tsx`
- `not-found.tsx`
- per-segment error boundaries

User copy must be:

> “We couldn’t load this page.”

4.2 Client Components

Show inline neutral messages:

> “Unable to continue. Please try again.”

Avoid popups unless absolutely required.
No red banners except for form validation.

4.3 Server Actions

All errors returned in canonical structure:

```
{
  ok: false,
  error: {
    code: "VALIDATION_ERROR" | "FORBIDDEN" | ...
    message: string
  }
}
```

The frontend **never receives stack traces or raw error objects**.

4.4 Database Level

If RLS denies access, the message is caught and mapped to:

> "You do not have permission to perform this action."

No SQL details ever surface.

4.5 Intelligence Engines

All engines wrap errors as:

```
{
  ok: false,
  engine: "MatchingEngine",
  version: "v1_2026_Q1",
  error: "ENGINE_ERROR"
}
```

Example:

> "We couldn't compute your match score."

---

5. Logging Architecture (Structured, Minimal, Safe)

Logging must:

- exclude PII
- exclude compensation
- exclude contact details
- exclude any sensitive fields
- use structured JSON
- use stable keys
- support correlation IDs
- log only what is necessary for diagnostics

5.1 Log Format

Every log line:

```
{
  "timestamp": "...",
  "level": "info" | "warn" | "error",
  "event": "server_action" | "engine_run" | "interaction" | "auth" |
"db",
  "action": "update_profile",
  "actor_role": "talent",
  "duration_ms": 128,
  "status": "ok",
  "error_code": null
}
```

If error:

```
{
  "timestamp": "...",
  "level": "error",
  "event": "db",
  "error_code": "RLS_DENIED",
  "action": "updateOpportunity",
  "actor_role": "brand_admin",
  "message": "RLS denied mutation",
  "duration_ms": 72
}
```

---

6. Observability Stack

Tailor Shift uses **three observability channels**:

6.1 Sentry (Frontend + Server)

Used for:

- runtime errors
- hydration issues
- unexpected SSR failures
- client exceptions
- slow server actions

Includes:

- session ID
- correlation ID
- error code

Never includes:

- names

- emails
- compensation
- preferences
- answers

6.2 Structured Logging (Server Actions + Engines)

Stored in:

- console (Vercel logs)
- log aggregation (future)

Used for:

- performance monitoring
- bottleneck detection
- incident analysis

6.3 Database Observability

Tiger Cloud-level insights:

- slow queries
- lock contention
- RLS denials
- error spikes

Used for tuning queries and debugging migrations.

---

7. Alerting

Alerts trigger only when:

- error rate > X% over 5 minutes
- RLS denials spike
- matching engine runs fail > Y times
- assessment engine fails > Z times
- server action latency spikes > threshold

Alerts go to:

- engineering Slack channel
- on-call system (future)

No alerts for expected validation errors.

---

8. Privacy & Compliance Rules

8.1 No PII in logs

Logs never include:

- names
- emails
- phone numbers
- compensation
- raw preferences
- assessment answers
- interaction notes

- free text fields

8.2 Data Minimisation

Only event metadata logged.

8.3 Immediate Sanitisation

Free-text form data or rejection reasons must **never** be logged.

8.4 Log Retention

- keep minimal logs
- 30–60 days default
- beyond that → aggregated, anonymized metrics

8.5 Exportability

Talent can export profile—but logs remain internal, anonymized.

---

9. Engine-Specific Observability

Each engine logs:

- version
- input vector length
- processing time
- error codes (if any)

Matching Engine

```
event: "engine_run",
engine: "MatchingEngine",
duration_ms: 211,
match_score: 78
```

Assessment Engine

```
engine: "AssessmentEngine",
scores: { service: 72, clienteling: 63, ... }
```

Learning Engine

```
engine: "LearningEngine",
recommended: 3
```

Projection Engine

```
engine: "ProjectionEngine",
output: { next_role: 4, next_tier: "T3" }
```

---

10. Testing — Canonical Requirements

10.1 Error Tests

- action rejection tests
- RLS denial tests
- insufficient permission tests
- malformed payload tests

10.2 Observability Tests

- verify logs appear
- verify logs are anonymized

10.3 UX Tests

- SSR error boundaries behave correctly
- inline errors appear cleanly
- no stack traces leaked

CHAPTER 21 — PERFORMANCE, CACHING & SCALABILITY (V5 CANON)

Global performance blueprint for a luxury-grade intelligence platform.

---

0. Purpose — Why Performance Matters in Tailor Shift

Luxury retail professionals and maisons expect:

- instant responsiveness,
- zero friction,
- calm, stable interactions,
- predictable loading patterns,
- no flicker, no chaos, no wait.

Performance is **UX**, **engine accuracy**, **trust**, and **credibility**.

This chapter defines the **exact performance budget**,
caching model, and **scalability architecture** for Tailor Shift.

---

1. Performance Budget (Canonical Targets)

Tailor Shift enforces strict performance budgets:

1.1 Frontend (SSR / App Router)

- SSR page generation: **< 150 ms** p50
- Talent Dashboard: **< 200 ms** p50

- Brand Dashboard: **< 200 ms** p50
- Opportunity listings: **< 180 ms**

1.2 Server Actions

- profile update: **< 120 ms**
- opportunity creation: **< 150 ms**
- assessment scoring: **< 200 ms**
- matching execution (single talent ↔ opportunity): **< 250 ms**

1.3 Database

- SELECT queries: **< 20 ms** p50
- complex joins (rare): **< 40 ms** p50
- JSONB filter queries: indexed → **< 50 ms** p50

1.4 Engines

- Matching Engine → **< 250 ms**
- Assessment Engine → **< 200 ms**
- Projection Engine → **< 300 ms**
- Learning Engine → **< 150 ms**

1.5 API Routes

- webhook calls: **< 80 ms**

1.6 SPA Actions (client components only)

- state transitions: **instant** (< 16 ms)

These targets shape architecture decisions.

2. Frontend Performance Architecture

2.1 Server Components Default

SSR ensures:

- no client-side waterfalls
- immediate hydration on minimal JS
- stable RBAC and RLS
- faster Time-to-Meaningful UI

2.2 Avoid Client Data Fetching

Forbidden:

- client-side SWR/React Query for protected data
- client-side requests to `/api/*` for internal data

2.3 Use Streaming for Large Views

Used in:

- opportunity lists
- search results
- brand store browsing

2.4 Suspense + Skeletons

Skeletons based on Tailor Shift design system:

- light grey blocks
- soft, neutral movement
- no shimmer animations

2.5 Component Memoisation

Reserved for:

- expensive charts
- capability summaries
- projection components

---

3. Caching Model (Deterministic & Safe)

Tailor Shift uses **three caching layers**, all safe under zero-trust constraints.

3.1 Layer 1 — Next.js Server-Side Cache

Used for:

- static marketing pages
- static assets
- public resources
- email templates
- scripts

Rules:

- public-only
- no sensitive data
- cache invalidation via route revalidation

3.2 Layer 2 — Engine Result Caching

Engines (matching, assessment, learning, projection) may cache:

- capability summaries
- stable match snapshots
- projection results

Stored in:

- transient memory
- or lightweight DB tables (matching_snapshots)

Cache Expiry:

- Talent profile change → flush match + projection
- assessment update → flush relevant engines
- opportunity change → flush matches for that opportunity

3.3 Layer 3 — Database Query Caching (Implicit)

Tiger Cloud automatically optimises:

- frequently used indexes
- hot JSONB paths
- repeated role-level filters
- division filters
- opportunity sorting queries

No manual caching here.
Indexes guarantee sub-20ms responses.

---

4. Scalability Strategy (V5)

Tailor Shift aims to scale globally with minimal infrastructure.

4.1 Horizontal Scaling (Frontend)

Vercel auto-scales SSR routes across regions.

No backend servers required.

4.2 Vertical Scaling (Database)

Tiger Cloud Postgres scales vertically and supports:

- read replicas (future)
- WAL-based scaling
- parallelised query execution

All indexing is optimised for relational + JSONB hybrid workloads.

4.3 Engine Scaling

Matching Engine

- runs lazily on demand
- supports batch recomputation via Edge Function
- match snapshots store stable results

Assessment Engine

- pure CPU-bound logic → trivial scaling

Learning Engine

- lightweight, synchronous

Projection Engine

- slightly heavier due to tier/role modeling
- still <300ms target

4.4 Concurrency Guarantees

Server Actions

- atomic DB operations
- optimistic concurrency via Postgres
- no deadlocks expected

Interactions

- unique key `(talent_id, opportunity_id)`
- ensures no duplicates

---

5. Query & Index Optimisation (V5)

Indexes defined in Chapter 17 ensure:

- fast filtering on divisions
- fast store-tier filtering
- fast role-level matching
- fast preference queries
- fast assessment lookups

Indexing Rules:

- never index large text fields
- index JSONB only with GIN
- always index foreign keys
- always index enums used for filtering
- avoid multi-column composite indexes unless proven necessary

---

6. Avoiding Performance Traps

6.1 Forbidden Patterns

- client-side heavy computation
- large client bundles
- unindexed JSONB queries
- GraphQL layers
- monolithic REST API
- polling on client side
- infinite scroll without pagination
- dynamic queries without prepared statements

6.2 Red Flags

- React context with large objects
- client-side filtering of large lists
- hydration mismatch warnings
- multiple server actions from one UI gesture

6.3 Strict Limits

- max 20 stores visible per page
- max 20 opportunities fetched initially
- max 100 talents returned to brands (sorted server-side)

More requires pagination or “load more.”

---

7. Load Testing & Benchmarking

7.1 Synthetic Profiles

- 100 simulated talents
- 20 store networks
- 50 opportunities
- engine benchmarks on each

7.2 Performance Tests

- match engine scaling tests
- search & filtering tests
- projection consistency checks

7.3 Acceptable Degradation

- p95 may go up to 2× p50 in heavy scenarios
- UX must still remain responsive

---

8. Monitoring Performance

Observability (Ch. 20) integrated with:

8.1 Metrics Tracked

- SSR duration
- server action duration
- DB query duration
- engine duration
- RLS denials
- interaction latency

8.2 Dashboards

- Vercel Analytics
- Tiger Cloud performance metrics
- Sentry performance tracking

8.3 Alerts

- slow queries >100ms
- SSR failures
- matching engine failures
- elevated RLS denials
- abnormal spike in 500 errors

---

9. Progressive Enhancement (Mobile-First)

9.1 Essential functionality must work without JS

- login
- signup
- profile update
- opportunity browsing

9.2 JS enhances only:

- assessment wizard
- learning module views
- multi-step forms
- filtering

---

10. CDN, Assets, & Image Strategy

10.1 Assets

- icons, logos, illustrations served from Vercel CDN
- compressed SVG for icons
- WebP for images

10.2 Fonts

- self-hosted or google fonts with preload
- Playfair Display + Manrope

10.3 No heavy renders

- avoid <img> for decorative content
- use vector assets when possible

CHAPTER 22 — SECURITY, PRIVACY & ZERO-TRUST (V5 CANON)

The formal security doctrine of Tailor Shift. Absolute discretion, strict boundaries, zero assumptions.

---

0. Purpose — Why Security Is Foundational

Tailor Shift handles:

- career data,
- capability signals,
- experience histories,
- professional preferences,
- store networks,
- opportunity structures,
- interactions between Talents and Houses.

Luxury environments value:

- confidentiality,
- trust,
- precision,
- stability,
- respect.

Security and privacy are therefore **core product values**, not an afterthought.

This chapter establishes **all non-negotiable rules** for:

- authentication
- authorization
- data access
- encryption
- confidentiality
- RBAC
- RLS
- privacy levels
- regulatory compliance

Tailor Shift V5 is designed to exceed expectations.

---

1. Security Philosophy — “Zero Trust with Elegant Edges”

Tailor Shift applies **zero-trust** at every layer:

- No user or system is trusted by default.
- No route is accessible without explicit permission.
- No table is readable without RLS approval.
- No sensitive field is exposed to the wrong audience.

UX remains **elegant, calm, and seamless**
— but security is **absolute** under the surface.

---

2. Identity & Access Control (Multi-Layer)

Security is enforced across **three independent layers**.

2.1 Layer 1 — Auth.js (Identity)

Handles:

- account creation
- login/logout
- session management
- OAuth
- password reset
- role assignment

All sessions use:

- HttpOnly, secure cookies
- server-side validation
- strict sameSite policy
- automatic token rotation

No JWT parsing in the client.

2.2 Layer 2 — Route Protection (Next.js Middleware)

Ensures unauthorized users cannot access pages.

Examples:

```
/talent/*       → allowed only to role=talent
/brand/*        → allowed only to role=brand_admin|recruiter
/admin/*        → internal only
```

Redirects unauthorized access to `/login`.

2.3 Layer 3 — Row-Level Security (Postgres)

The ultimate security boundary.
Even if the frontend and middleware fail, the database prevents illegal access.

RLS rules ensure:

- a Talent cannot read another Talent
- a House cannot read another House
- no Brand can see private Talent fields
- interactions are visible only to their participants
- no mutation happens unless the user owns the data

This is the strongest guarantee in the system.

---

3. Data Ownership — Formal Rules

3.1 Talent Owns

- profile
- preferences
- experience blocks
- assessment results
- capability summary
- compensation profile
- learning history
- projections

3.2 Brand Owns

- brand profile
- stores
- opportunities
- brand admin identities
- interactions they originated

3.3 Tailor Shift Owns

- system events
- engine metadata
- anonymized logs

Nothing else.

---

4. Sensitive Fields — Protection Rules

4.1 Always Private (Never Exposed)

- assessment answers
- compensation_profile (Talent)
- preferences (Talent)
- contact details (masked pre-acceptance)
- Talent’s learning history
- Talent’s projections
- Talent’s assessment insights

4.2 Exposed Only After Consent

Upon **mutual acceptance** (interaction = `accepted`):

- email
- phone (if provided)
- preferred contact method

4.3 Exposed in Public View Only

- role level
- divisions
- years of experience
- location (city/region)
- capability summary (neutral, aggregated)
- experience blocks (clean, non-sensitive)

---

5. Database Security (PostgreSQL-Level)

5.1 RLS Enabled for ALL Domain Tables

- talents
- brands
- stores
- opportunities
- experience_blocks
- assessments
- interactions
- learning_progress

5.2 Default Deny

Every table is:

```
ALTER TABLE X ENABLE ROW LEVEL SECURITY;
ALTER TABLE X FORCE ROW LEVEL SECURITY;
```

5.3 No Superuser Shortcuts

Even platform Admins must use controlled views.

---

6. Encryption Policy

6.1 At Rest

Tiger Cloud PostgreSQL encrypts all data at rest, including:

- Talent profiles
- Experience blocks
- Opportunities
- Compensation data

6.2 In Transit

All connections use:

- HTTPS
- TLS 1.2 or higher
- secure WebSockets (if used)

6.3 Sensitive Field Encryption

Certain fields may be encrypted with Postgres `pgcrypto`:

- compensation_profile

- sensitive brand documents (future)

---

7. Frontend Security Rules

7.1 Forbidden

- storing tokens in localStorage
- exposing session data to client side
- direct fetch to protected resources from client
- embedding raw API URLs in client code
- using dangerouslySetInnerHTML on user data
- rendering sensitive fields in Client Components

7.2 Required

- HttpOnly secure cookies
- minimal hydration
- sanitized form inputs
- defensive coding patterns

---

8. Server Action Security

All server actions must:

- validate session
- validate role
- validate inputs

- check RLS success
- never log sensitive content
- return structured errors

Example:

```
if (session.role !== "talent") {
  return { ok: false, error: { code: "FORBIDDEN", message: "Not
allowed." } };
}
```

---

9. Interaction Security (Talent ↔ Brand)

Defined in Ch. 13, but summarised here:

9.1 Allowed to Brand (before acceptance)

- Talent public view only
- capability summary
- years of experience
- division experience
- tier exposure

9.2 Forbidden to Brand (before acceptance)

- Talent preferences
- assessment answers
- compensation
- projections
- learning history

- contact details

9.3 Allowed After Acceptance

- name
- email
- phone

Never anything else.

---

10. Audit Logging

All privileged actions generate logs:

- login
- logout
- profile update
- opportunity creation
- interaction acceptance
- server action failures
- RLS denials

Logs contain:

- timestamp
- event
- action
- duration
- error code

Never contain:

- names
- emails
- compensation
- preferences
- sensitive free-text

---

11. GDPR Compliance

11.1 Right to Be Forgotten

When Talent requests deletion:

1. `auth.users` row deleted
2. dependent rows cascade-delete
3. Talent fields anonymized in aggregated datasets
4. interactions remain, but Talent ID replaced with `null` or `anonymized_id`

11.2 Right to Access

Talent can export:

- their profile
- experience blocks
- assessment summaries
- preferences
- learning progress

Brand can export its:

- stores
- opportunities
- admin accounts

11.3 Data Minimisation

No data collected unless:

- required for matching
- required for Talent display
- required for interaction
- required for learning or projection
- compliant with Talent consent

---

12. EU AI Act Compliance

Tailor Shift falls under **low-risk AI classification** because:

- all engines are deterministic
- no ML models infer sensitive attributes
- no automated selection decisions
- explanations accompany all scores
- Talent has full transparency
- decisions require **human review**
- no opaque ranking logic
- no psychometric profiling

---

13. Penetration Resistance

Tests required:

- SQL injection → impossible behind RLS
- XSS → all inputs sanitised
- CSRF → server actions protected by same-site cookies
- OAuth misconfiguration → denied at callback
- Replay attacks → session rotation
- Brute force → rate limiting

Prevented by design:

- privilege escalation
- lateral movement between Tenants
- direct DB access
- unauthorized brand access

---

14. Security Hardening Checklist

Mandatory before production:

- RLS active on all tables
- All server actions role-validated
- No raw SQL in client
- All form input sanitised
- No sensitive field rendered in client

- Session cookies secure/HttpOnly
- SSR-only data rendering
- Error boundaries active
- Logging anonymized
- Backup + recovery tested

CHAPTER 23 — DEVOPS, CI/CD & DEPLOYMENT (V5 CANON)

Deterministic, safe, predictable DevOps for a global luxury intelligence platform.

---

0. Purpose — Why DevOps Matters in Tailor Shift

Tailor Shift’s users (luxury houses & high-end professionals) expect:

- **no downtime**,
- **zero data inconsistency**,
- **fast releases**,
- **safe migrations**,
- **rigorous versioning**,
- **calm, predictable deployments**,
- **bulletproof compliance**.

DevOps is the backbone that prevents:

- regressions,

- database drift,
- RLS breaches,
- engine drift,
- UX inconsistency,
- build instability.

This chapter establishes **the official V5 DevOps standards**.

---

1. Environment Strategy

Tailor Shift uses **three fully isolated environments**:

```
local → staging → production
```

Each environment has its own:

- Tiger Cloud Postgres instance
- Auth.js configuration
- Vercel deployment
- secrets store

1.1 Separation of Concerns

- Local: developer sandbox
- Staging: QA, designer review, client demos
- Production: end-user traffic

1.2 Forbidden

- no staging → production connection

- no production DB queries from local
- no production debugging via direct SQL

---

2. Secrets Management

All secrets stored in:

- Vercel environment variables (staging/prod)
- `.env.local` (ignored in Git)

Never Stored In:

- client code
- git history
- logs
- public assets

Critical Secrets

- Auth.js secrets
- Tiger Cloud database URL
- encryption keys (pgcrypto)
- service role key (backend-only)

Service role key NEVER ships to browser bundles.

---

3. CI/CD Pipeline (GitHub Actions)

CI/CD enforces correctness before deployment.

3.1 Trigger Conditions

- pull request open/updated
- merge into `main` or `staging`
- tagged release

3.2 CI Phases (Canonical)

Phase 1 — Install & Type Check

```
npm install
npm run type-check
```

Fail if:

- incorrect types
- missing props
- invalid form data structures

Phase 2 — Lint

```
npm run lint
```

Fail if:

- coding standards broken
- unused variables
- unreachable code

Phase 3 — Test Engines

Unit tests for:

- Matching

- Assessment
- Learning
- Projection
- Interaction state machine

Targets:

- deterministic
- consistent
- explainable

Phase 4 — Integration Tests

Integration scenarios test:

- RLS access boundaries
- server actions
- Talent onboarding
- Brand opportunity creation
- Interaction acceptance

Phase 5 — Build

```
npm run build
```

Build must:

- SSR all pages
- tree-shake client components
- validate server actions
- render all layouts correctly

Phase 6 — Preview Deployment (Staging)

Push merge commit → staging deploy.

Phase 7 — QA Gates

QA checks:

- UX correctness
- engine outputs
- performance metrics
- error boundaries
- routing

---

4. Deployment Rules

4.1 Staging Deployment

Automatic on merge into `staging`.

4.2 Production Deployment

Manual promotion from staging after:

- QA approval
- no regression
- migration validated
- RLS tests passing
- performance checks stable
- Sentry clean

4.3 Rollbacks

Every deployment is fully reversible.

Procedure:

1. revert to previous Vercel build
2. apply previous DB schema migration (down migration)
3. warm caches
4. verify SSR and engines

---

5. Database Migrations

Migrations follow strict discipline.

5.1 Migration Workflow

Local Development

```
supabase db push
```

Generates migration SQL files.

Commit migrations

All migration files MUST be checked in:

```
/migrations/YYYYMMDDHHMM_added_feature.sql
```

Staging Deployment

```
supabase db push --project-ref <staging>
```

Production Deployment

```
supabase db push --project-ref <prod>
```

5.2 Migration Rules

- 1 migration = 1 schema change

- Never modify existing migration files
- Never create manual tables via dashboard
- Always test RLS rules after migration
- Always test opportunity CRUD after schema updates
- Always test Talent onboarding

5.3 Migration Reviews

Every migration must be:

- reviewed for breaking changes
- tested on staging
- verified with data

5.4 Reversible

Down migrations required for production.

---

6. Engine Versioning & Deployment

Each engine (Matching, Assessment, Learning, Projection) must declare:

```
engine_version: "EngineName_vN_YYYY_QX"
```

6.1 Versioning Triggers

Update version when:

- weights change
- scoring logic changes
- new dimension added
- insight rules updated

6.2 Canaries & Rollouts

- engine tested on synthetic dataset
- rollout first in staging
- engine selectors can route % of users (future optional)
- rollback by re-selecting previous version

---

7. Monitoring & Alerting

7.1 Monitoring Tools

- Vercel Analytics
- Sentry
- Tiger Cloud logs
- Custom engine logs

7.2 Monitored Metrics

- SSR latency
- server action latency
- DB query latency
- RLS denials
- matching engine failures
- assessment engine failures
- error spikes
- interaction failure rate

7.3 Alert Thresholds

Alerts triggered when:

- error rate > 2% for 5 minutes
- SSR latency > 400ms p95
- RLS denials spike unexpectedly
- engine failure rate > 10 occurrences/hour
- DB slow queries > threshold

---

8. Backup & Recovery

8.1 Automated Backups

- Tiger Cloud hourly backups
- retained for 14–30 days

8.2 Recovery Procedure

1. restore staging from backup
2. verify data integrity
3. rehearse full recovery
4. only restore production if disaster-level incident

8.3 Disaster Classes

- accidental data deletion
- migration failure
- corrupted schema
- RLS misconfiguration

- cloud provider outage

---

9. Deployment Safety Nets

9.1 Feature Flags

Used for:

- new UX
- new engine versions
- experimental components
- new opportunity flows

9.2 Kill Switches

Instantly disable:

- engine execution
- opportunity posting
- interaction initiation

---

10. Recommended Branching Strategy

main

Always production-ready.
Manual deploy → production.

staging

Integration branch.
Auto deploy → staging.

feature/ branches*

One per story/task.

release/ branches*

Optional for large versions.

---

11. Local Dev Standards

11.1 Required Tools

- Node LTS
- Supabase CLI
- psql
- Typescript
- Vercel CLI (optional)

11.2 Local DB

```
supabase start
```

11.3 Seed Data

Local seeds must use sanitized, fictional data.

CHAPTER 24 — TESTING & QUALITY ASSURANCE (V5 CANON)

Rigorous, automated, privacy-compliant QA for a global luxury talent platform.

---

0. Purpose — Why QA Is Essential

Luxury retail professionals and houses expect **polish, stability, and trust**.
Tailor Shift must never:

- expose sensitive fields,
- break RLS,
- regress in UI,
- miscompute intelligence outputs,
- display inconsistent behaviour,
- produce misleading match/assessment/projection results.

This chapter establishes:

- **the full testing matrix**,
- **testing tools**,
- **coverage requirements**,
- **QA gates for staging & production**,
- **how engines must be validated**.

---

1. QA Philosophy — "Deterministic Quality"

All logic in Tailor Shift is:

- deterministic,
- explainable,
- domain-based,
- privacy-sensitive,
- rule-driven.

Therefore QA must verify:

1. **Consistency** (same input → same output)
2. **Explainability** (all results interpretable)
3. **Correctness** (ontology-aligned)
4. **Security** (no access leaks)
5. **Privacy** (no sensitive field exposure)
6. **Performance** (fast interactions)
7. **Integrity** (schema, RLS, engines)

---

2. Types of Tests Required

Tailor Shift requires **nine test categories**.

2.1 Type 1 — Unit Tests (Logic-Level)

Highly focused tests for:

- Matching Engine
- Assessment Engine
- Learning Engine
- Projection Engine
- Experience Block parsers
- Capability calculators
- Field validators

Goal: verify pure logic.

Example:

```
expect(RoleFit(L3, L4)).toEqual(85)
```

2.2 Type 2 — RLS Tests (Critical)

Tests verifying Postgres Row-Level Security rules.

These tests validate:

- a Talent cannot read another Talent
- a Brand cannot mutate another Brand
- interactions visible only to involved parties
- opportunities visible only to Brand owners

Failing any RLS test = **blocker**.

2.3 Type 3 — Server Action Tests

Validate:

- RBAC enforcement
- correct inputs
- error typing
- expected side effects
- expected RLS behaviour

Example:

```
expect(updateTalentProfileAction(…)).toThrow(FORBIDDEN)
```

2.4 Type 4 — Integration Tests

End-to-end tests covering entire flows:

- Talent onboarding
- Brand onboarding
- Opportunity creation
- Assessment completion
- Interaction creation → acceptance
- Matching computation
- Projection after assessment

These run against a **local DB with seed data**.

2.5 Type 5 — UI Snapshot Tests

SSR snapshots to detect regressions in:

- layout
- card components
- dashboard modules
- opportunity cards
- talent cards
- assessment pages

Snapshots ensure visual consistency across releases.

2.6 Type 6 — Accessibility Tests

WCAG 2.1 AA checks:

- semantic HTML

- visible focus rings
- form label associations
- color contrast
- keyboard-only navigation

Accessibility is mandatory for luxury profession standards.

2.7 Type 7 — Performance Tests

Validate performance budgets (Ch.21):

- SSR < 150ms
- matching < 250ms
- projection < 300ms
- RLS queries < 20ms

Failing performance budgets is a **release blocker**.

2.8 Type 8 — Regression Tests

Test entire system after schema changes, including:

- updated migrations
- updated engine weights
- new components
- new server actions

Everything must remain consistent.

2.9 Type 9 — Security Tests

Penetration-style testing:

- SQL injection
- RLS bypass
- privilege escalation
- session hijack
- CSRF
- XSS
- OAuth misconfiguration
- forbidden access attempts

Security tests must pass before any production release.

---

3. QA Infrastructure

3.1 Test Environments

- local (developer)
- CI (GitHub Actions)
- staging (full regression suite)

3.2 Test Data

seeded with **synthetic luxury retail data**, including:

- 50 Talents
- 10 Brands
- 20 Stores

- 30 Opportunities
- variety of divisions, tiers, levels

3.3 Reset Strategy

Local reset:

```
supabase db reset
supabase db push
npm run seed
```

---

4. Required Coverage

Minimum required coverage:

- **Logic (engines): ≥ 90%**
- **Server actions: ≥ 85%**
- **RLS: 100% coverage of critical paths**
- **UI snapshots: key 20 screens**
- **Accessibility: 100% must pass lint checks**

Coverage is a **non-negotiable requirement** for production.

---

5. QA Gates (Staging → Production)

A deployment **cannot** go to production unless:

5.1 All CI checks green

- lint
- type-check

- unit tests
- integration tests
- engine tests
- build

5.2 Staging performance under thresholds

- SSR < 150–200ms
- server actions < 150ms
- match engine stable

5.3 No uncaught errors or console warnings

5.4 RLS tests all pass

5.5 Manual QA checks

- onboarding flows
- brand flows
- interaction acceptance
- match breakdown clarity
- privacy boundaries respected

Any violation = **release blocked**.

---

6. Manual QA Checklist (Luxury-Specific)

Because Tailor Shift serves luxury clients, manual QA includes experiential checks:

- typography consistency

- spacing rhythm verification
- calm, premium loading states
- no visual noise
- stable layouts on mobile
- no jumpy transitions
- no overlapping components
- card clarity (no clutter)

These checks ensure luxury brand standards.

---

7. Engine Validation Framework

Each engine must have a **fixed validation suite**, run before any release.

7.1 Matching Engine

- 20 synthetic profiles × 10 opportunities
- identical outputs for identical inputs
- correct compensation alignment
- correct division weighting
- correct tier transitions

7.2 Assessment Engine

- same answers → same capability summary
- inferential adjustments tested
- division and tier adjustments validated

7.3 Learning Engine

- ranking stable
- gap detection consistent
- trajectory mapping correct

7.4 Projection Engine

- role-level projections correct
- tier projections correct
- timeline ranges within bounds

---

8. Error & Logging Validation

Every release must verify:

- error pages render correctly
- user-facing copy is neutral and calm
- logs contain NO PII
- failed actions surface correct error codes

---

9. QA Ownership & Roles

9.1 Engineering

- write all tests
- maintain engine correctness
- maintain RLS rules
- guarantee zero violations

9.2 Product

- validates UX flows
- ensures luxury UX fidelity
- validates opportunity templates
- maintains MCS consistency

9.3 QA Team (or delegated engineer)

- runs full regression suite on staging
- verifies interactions & privacy boundaries
- signs off before production

No production release without explicit approval.

CHAPTER 25 — DEPLOYMENT ENVIRONMENTS & RELEASE MANAGEMENT (V5 CANON)

Clear, disciplined, fully-isolated environments with controlled, luxury-grade release flows.

---

0. Purpose — Why This Chapter Exists

A luxury-grade platform must never:

- break in production,
- leak data between environments,
- regress silently,
- expose unfinished work to users,

- suffer database drift,
- deploy untested features,
- roll out chaotic releases.

Tailor Shift V5 enforces **strict environment isolation**,
predictable releases, and **deployments with consent**.

---

1. Environment Model (V5)

Tailor Shift operates with **three fully isolated environments**:

```
local → staging → production
```

All environments have **separate**:

- database
- secrets
- authentication configuration
- storage
- analytics
- logs
- caching

There is **zero shared state**.

---

2. Environment Definitions

2.1 Local (Developer Sandbox)

Purpose:

- feature development
- debugging
- schema evolution
- engine calibration
- local seed data

Characteristics:

- ephemeral
- seeded with synthetic luxury-retail data
- fast iteration
- logs verbose
- hot reload

Forbidden:

- using production secrets
- using production/staging data

2.2 Staging (QA & Validation)

Purpose:

- integration testing
- RLS verification
- engine validation
- matching/assessment scenarios
- UX validation

- stakeholder previews

Characteristics:

- stable but non-production
- seeded with curated demo dataset
- closely mirrors production infrastructure
- Sentry active
- performance metrics tracked

Forbidden:

- real user data
- experimental debugging with real accounts
- unreviewed migrations

2.3 Production (Live Platform)

Purpose:

- serving real Talents and Houses
- powering intelligence for real workflows
- high availability, low latency

Characteristics:

- high stability
- strict monitoring
- full observability
- complete RLS protection

- no development tools

Forbidden:

- schema editing outside migration pipeline
- exposing internal logs
- pushing unvalidated code
- debugging by bypassing RLS
- using console queries to inspect user data

---

3. Release Pipeline (Mandatory)

Tailor Shift uses a **controlled release process**:

```
feature/* → staging → production (manual promotion)
```

3.1 Step 1 — Develop on Feature Branches

- each feature = 1 branch
- always up-to-date with `staging`
- commits must reference story/spec

3.2 Step 2 — Merge to Staging → Auto Deploy to Staging

CI must:

- install dependencies

- type-check
- lint
- test engines
- run RLS tests
- run integration suite
- build the app

Only if all checks pass → deploy to staging.

3.3 Step 3 — Staging QA Gate

Manual testing on staging validates:

- UX consistency
- full Talent onboarding
- full Brand onboarding
- opportunity creation
- matching breakdown
- interaction flows
- dashboard rendering
- privacy boundaries
- engine correctness
- performance budgets
- no UI regressions
- no console errors

If ANY check fails → fix on feature branch → redeploy to staging.

3.4 Step 4 — Promotion to Production (Manual)

Production deploy must be:

- manual
- deliberate
- approved

Requirements:

- staging green
- migrations validated
- no RLS failures
- performance stable
- logs clean
- UX verified
- interactions functional

Promotion performed via Vercel “Promote to Production”.

3.5 Step 5 — Post-Deployment Verification

Metrics to check:

- SSR latency (p50 / p95)
- server actions
- matching engine errors
- RLS denials

- Sentry exceptions
- interaction acceptance flow
- opportunity listing

This verification must be performed within 10–20 minutes post-deploy.

---

4. Hotfix Process (Critical)

Hotfixes follow a **separate emergency path**:

```
production → hotfix/* → staging hotfix → production
```

Steps:

1. Create branch `hotfix/<issue>` from **production**.
2. Apply minimal fix.
3. Deploy to **staging** only.
4. QA smoke test.
5. Promote to **production**.
6. Merge hotfix back into `main` and `staging`.

Forbidden:

- deploying hotfix without staging validation
- “quick fixes” that bypass engine or RLS rules
- using console edits in production

---

5. Versioning & Releases

Version format:

```
TailorShift_V<major>.<minor>.<patch> (YYYY-MM-DD)
```

Major versions

- structural changes
- new engines
- redesigned UX
- new schema

Minor versions

- improvements
- new components
- refined matching logic
- performance optimizations

Patch versions

- bug fixes
- copy corrections
- UI adjustments

All changes must correspond to Spec amendments.

---

6. Rollback Policy

Rollback must be:

- fast

- safe
- predictable
- reversible

Backend rollback:

1. promote previous production build
2. apply “down migration”
3. validate RLS
4. validate core flows

Frontend rollback:

Instant via Vercel previous deployment.

Rollback must never result in partial schema mismatches.

---

7. Release Cadence

Suggested cadence:

- Weekly minor releases
- Monthly patch releases
- Quarterly major releases

Urgent fixes:

- hotfix anytime (never without staging validation)

---

8. Deployment Compliance (Luxury Industry Expectations)

Every deployment must satisfy:

- high reliability
- controlled pacing
- respectful UX
- zero regressions
- complete auditability
- GDPR compliance
- low-risk AI classification compliance
- no sensitive data transfer
- structured logging
- engine version stability

This ensures Tailor Shift remains credible to global maisons.

CHAPTER 26 — ENVIRONMENT VARIABLES & SECRET MANAGEMENT (V5 CANON)

Strict, minimal, safe, and auditable environment configuration for Tailor Shift.

---

0. Purpose — Why Environment & Secret Management Matters

Luxury retail systems require:

- zero tolerance for leaks,
- controlled execution contexts,
- deterministic configuration across environments,
- perfect consistency between staging and production,
- tight separation between local/staging/production.

Mismanaged secrets =
data leaks,
broken auth,
RLS bypass,
GDPR violation,
reputational damage.

This chapter protects Tailor Shift against those risks.

---

1. Principles — The Tailor Shift Secret Doctrine

1. **Minimal Surface**
   Only declare what is strictly necessary.
2. **No Secrets in Client Bundles**
   EVER.
   All sensitive env vars MUST be server-side only.
3. **No Secrets in Git**
   `.env.local` is `.gitignore`d.
   No exceptions.
4. **One Source of Truth per Environment**
   - Local: `.env.local`
   - Staging: Vercel env vars
   - Production: Vercel env vars

5. **Immutable on Deployment**
   Changing secrets forces a redeploy.

6. **Never Log Secret Values**
   Logging secrets is strictly forbidden.

7. **Separation of Duties**
   A secret belongs to a single subsystem (Db/Auth/etc.).
   Never reuse a secret across systems.

8. **Rotation Capability**
   Secrets must be rotatable with zero downtime.

---

2. Environment Variable Categories

Environment variables fall into 4 groups:

A. Authentication variables

B. Database variables

C. Runtime variables

D. Infrastructure variables

We specify each below.

---

3. Authentication Variables (Auth.js)

These control identity, sessions, OAuth flows.

3.1 Required Variables

```
AUTH_SECRET=...
AUTH_URL=https://tailorshift.co
NEXTAUTH_URL=https://tailorshift.co      (legacy alias)
```

3.2 OAuth Providers (Optional in V5)

Google OAuth

```
GOOGLE_CLIENT_ID=...
GOOGLE_CLIENT_SECRET=...
```

LinkedIn OAuth

```
LINKEDIN_CLIENT_ID=...
LINKEDIN_CLIENT_SECRET=...
```

Apple OAuth (V5.1+)

```
APPLE_CLIENT_ID=...
APPLE_TEAM_ID=...
APPLE_KEY_ID=...
APPLE_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n...\n-----END
PRIVATE KEY-----"
```

Important:
OAuth private keys must be multi-line secrets stored only in Vercel's encrypted storage.
NEVER in `.env.local`.

---

4. Database Variables (Tiger Cloud PostgreSQL)

4.1 Required Variables

```
DATABASE_URL=postgresql://user:password@host/db
DIRECT_URL=postgresql://user:password@host/db
```

4.2 Do not expose

Forbidden:

- SERVICE ROLE KEY in browser
- DATABASE_URL in client code
- passwords in logs

4.3 Proper usage pattern in code

Only server components, server actions, or edge functions may touch `DATABASE_URL`.

Example:

```
import { db } from "@/lib/db"; // server-side only
```

Never:

```
fetch(process.env.DATABASE_URL) // forbidden
```

---

5. Runtime Application Variables

These variables tune behaviour across environments.

5.1 Node Environment

```
NODE_ENV=production | development
```

5.2 App Environment

```
APP_ENV=local | staging | production
```

5.3 Base URLs

```
NEXT_PUBLIC_APP_URL=https://tailorshift.co
NEXT_PUBLIC_STAGING_URL=https://staging.tailorshift.co
```

Note:
Anything with `NEXT_PUBLIC_` becomes available in the client bundle.
Must be non-sensitive.

---

6. Intelligence Engine Variables

Engine configs are versioned inside the codebase, **not** environment variables.
BUT engine toggles can be declared as non-sensitive env vars:

```
ENGINE_MATCHING_VERSION=v1_2026_Q1
ENGINE_ASSESSMENT_VERSION=v1_2026_Q1
ENGINE_LEARNING_VERSION=v1_2026_Q1
ENGINE_PROJECTION_VERSION=v1_2026_Q1
```

These guide:

- explainability logs
- version pinning in CI
- rollouts

They do *not* control internal engine weights, which live in code.

---

7. Logging & Observability Variables

```
SENTRY_DSN=...
SENTRY_ENVIRONMENT=local|staging|production
SENTRY_TRACES_SAMPLE_RATE=0.2
SENTRY_PROFILES_SAMPLE_RATE=0.1
```

All must be server-only.
Client logs are proxied safely by Sentry without exposing tokens.

---

8. Email Provider Variables (Optional V5, Mandatory V6)

If using transactional email:

```
EMAIL_PROVIDER=smtp|postmark|sendgrid
EMAIL_API_KEY=...
EMAIL_FROM_ADDRESS=no-reply@tailorshift.co
```

---

9. Infrastructure Variables

9.1 Vercel

```
VERCEL_ENV=development | preview | production
VERCEL_REGION=iad1 | cdg1 | ...
```

9.2 Analytics (Optional)

```
TAILORSHIFT_ANALYTICS_TOKEN=...
```

Must be anonymized.

---

10. Forbidden Patterns (Critical)

❌ Secrets in client-side code

If any variable starting with a normal name (no NEXT_PUBLIC) is imported client-side → BLOCKER.

❌ Service Role Keys in client code

Severe security breach.

❌ Putting secrets inside JSON files

e.g., `config.json` → forbidden.

❌ Storing secrets in GitHub Actions plaintext

Must use GH environment secrets.

❌ Mixing production/staging/local URLs

Causes leaked sessions. Forbidden.

❌ Trying to access DATABASE_URL from browser

Should be impossible.

---

11. Secret Rotation Procedure

For critical secrets (Auth Secret, Database password, OAuth private keys):

1. Add new secrets to:
   - Vercel staging
   - Vercel production
   - `.env.local` (developer machines)
2. Redeploy staging
3. Test full flows:
   - login
   - signup
   - password reset
   - brand onboarding
   - interaction acceptance
4. Redeploy production
5. Revoke old secrets

Rotation must never break sessions unexpectedly.

---

12. Environment-Specific Config Behaviour

12.1 Local (.env.local)

- can include mild debugging toggles
- must never contain production DB credentials

- credentials are fictional/synthetic

12.2 Staging

- synthetic data only
- cleaned logs
- full RLS enabled
- engines fully active

12.3 Production

- no debug toggles
- no verbose logging
- strict mode enabled
- sensitive logs disabled

---

13. Validation & CI Checks

A CI step validates the environment configuration:

- required secrets must exist
- no missing keys
- no unscoped NEXT_PUBLIC secrets
- no duplication
- no misnamed secrets
- engine versions aligned
- Auth.js config validated

Why?
Incorrect environment config = silent failures, regressions, RLS misfires, inconsistent SSR, incorrect cookies.

CHAPTER 27 — INFRASTRUCTURE-AS-CODE & AUTOMATION (V5 CANON)

Predictable, reproducible, zero-drift infrastructure for a global luxury availability platform.

---

0. Purpose — Why IaC Is Mandatory

Tailor Shift V5 requires:

- zero configuration drift
- consistent environment parity
- auditable infra changes
- deterministic deployments
- safe migrations
- quick recovery
- privacy and compliance guarantees

Infrastructure-as-Code (IaC) ensures:

- **every infrastructural element is versioned**,
- **reproducible environments**,
- **human errors eliminated**,
- **automated provisioning**,
- **immutable deployments**.

Nothing in the system may be configured manually without IaC or a matching migration.

---

1. Infrastructure Stack Overview

Tailor Shift uses a **minimal yet robust cloud footprint**, composed of:

1. **Vercel**
    - serverless Next.js runtime
    - global CDN
    - automatic scaling
2. **Tiger Cloud PostgreSQL**
    - managed database
    - RLS
    - daily/hourly backups
    - encryption at rest
3. **Auth.js**
    - identity & session management
    - OAuth provider integration
4. **Sentry**
    - logging
    - performance
    - error monitoring
5. **GitHub Actions**
    - CI/CD pipeline
    - automated testing

    - migration enforcement

6. **Storage (future V6)**
    - for brand materials
    - limited, encrypted

This entire footprint must be defined & governed through automation.

---

2. IaC Philosophy — “Code or It Doesn’t Exist”

All infrastructure must be:

- declared in code,
- versioned in Git,
- deployed via CI/CD,
- reproducible in local/staging/prod,
- audited automatically.

Manual dashboard clicks are forbidden unless explicitly mirrored in code.

---

3. What Is Managed via IaC (V5)?

Must be IaC:

- Vercel project configuration
- environment variables
- build & deploy settings

- routing rules
- Sentry integration
- DB migrations
- seed data generation
- monitoring alerts
- cron / scheduled tasks
- Edge Functions (if any)

Not IaC (for now):

- OAuth provider configuration (managed externally)
- Tiger Cloud instance creation (initial manual)
- Backups (managed by platform)

---

4. IaC Tools Used in V5

4.1 Terraform (recommended)

Used to codify:

- Vercel projects
- environment configuration
- domain settings
- DNS records
- Sentry projects
- scheduled functions
- environment variables

4.2 Supabase CLI / SQL Migrations

Used to codify:

- DB schema
- RLS policies
- triggers
- seed pipeline
- extensions

4.3 GitHub Actions

Used to orchestrate:

- migrations
- deploys
- engine tests
- smoke tests

4.4 Optional Future: Pulumi

Not required for V5 but fits well for multi-cloud.

---

5. Git Repository Structure (IaC Section)

Tailor Shift repo must include:

```
/infra
  /vercel
    main.tf
    variables.tf
    outputs.tf

  /monitoring
    sentry.tf
    alerts.tf

  /env
    prod.env
    staging.env
    README.md

/migrations
  <timestamp>_migration.sql

/scripts
  seed.ts
  seed.json

.github/workflows
  ci.yml
  deploy.yml
  migrations.yml
```

---

6. Infrastructure Lifecycle

6.1 Provision

Use Terraform to apply:

```
terraform init
terraform plan
terraform apply
```

Provision:

- project
- env vars
- CDN domains
- build settings

- redirects
- Sentry project

6.2 Update

Changes must be added to code, not dashboards.
Example:

- new env var
- new domain
- new scheduled function

6.3 Destroy

Only allowed for local/test environments.

---

7. Automated Tasks & Scheduled Jobs

7.1 Daily Jobs

- match snapshot cleanup
- engine version rollups
- stale interaction cleanup

7.2 Weekly Jobs

- unused brand assets cleanup
- capability statistical review (anonymized)

7.3 Monthly Jobs

- backup integrity test

- RLS enforcement test

7.4 Tools

- Vercel Cron Jobs (API routes)
- Edge Functions (future)
- GitHub scheduled workflows

---

8. Zero-Drift Enforcement

8.1 Drift Detection

Terraform plan must run on CI daily:

- detect manual changes in Vercel
- detect missing env vars
- detect mismatched settings

8.2 Drift Alerts

Triggered if:

- resource changed manually
- env var missing
- DNS updated outside IaC

8.3 Drift Remediation

- apply IaC definitions
- or document intended change and update IaC

Every environment must match its IaC plan exactly.

9. Security in IaC

9.1 Secrets NEVER in Terraform files

Instead:

```
variable "auth_secret" {
  type = string
  sensitive = true
}
```

Values injected via:

- Vercel encrypted secrets
- GitHub Actions secrets

9.2 Sensitive Outputs Hidden

Terraform must never output secret values.

9.3 Lock State

Terraform state stored in secured backend (S3/GCS-equivalent, encrypted).

9.4 RLS Policies in SQL

Never modify RLS by hand in dashboard.
RLS changes require:

1. SQL migration
2. CI tests
3. staging validation

10. Continuous Verification (IaC QA)

Every PR must:

- run `terraform validate`
- run `supabase db lint`
- run RLS tests
- run schema comparisons

If IaC fails → PR blocked.

---

11. Multi-Environment Strategy (IaC)

Terraform must allow deploying multiple isolated stacks:

```
workspace "staging" {
  ...
}
workspace "production" {
  ...
}
```

Each workspace has:

- different DB URLs
- different secrets
- different domain
- same schema, engines, and config

---

12. Disaster Recovery (IaC)

IaC ensures **rapid reconstruction** of:

- Vercel config
- DNS
- Route configs
- Environment variables

Combined with:

- DB backups (Ch. 29)
- restore procedures

Tailor Shift can be restored fully within hours.

---

13. Luxury-Grade Operational Discipline

In luxury contexts, downtime is unacceptable.

IaC ensures:

- predictable scaling
- reliable deployments
- no forgotten manual tweaks
- absolute symmetry across envs
- clear audit trails
- rapid iteration without chaos

CHAPTER 28 — ANALYTICS, METRICS & KPI FRAMEWORK (V5 CANON)

Minimal, privacy-safe, luxury-grade analytics designed for product relevance, not surveillance.

---

0. Purpose — Why Analytics Must Be Carefully Designed

Luxury professionals and houses demand:

- discretion
- low noise
- zero invasive tracking
- complete transparency
- GDPR compliance
- clean audit trails

Tailor Shift analytics must respect:

- user trust
- industry expectations
- legal constraints
- the privacy-by-design architecture

The Analytics framework here is **minimal, respectful, product-focused**, and aligned with the intelligence layer.

---

1. Analytics Philosophy — “Measure Value, Not People”

Tailor Shift collects:

- product usage patterns (aggregate)

- engine performance
- matching effectiveness
- reliability metrics
- UX friction

Tailor Shift **never** collects:

- behavioural profiling
- demographic inference
- keystroke data
- cursor tracking
- personal browsing habits
- cross-site identifiers

Analytics must inform **product quality** and **matching accuracy**, not invade user privacy.

---

2. What Tailor Shift Tracks (Allowed)

Analytics are grouped into **five canonical categories**:

2.1 Product Usage Metrics (Anonymous & Aggregate)

- page load counts
- dashboard load counts
- module opens (learning, assessment)
- assessment started / completed
- opportunity detail views

- profile completion rates
- store creation events
- opportunity creation events
- interaction requests (count only)

These are **aggregate** and **never tied to a real identity**.

Data stored as:

```
{ event: "assessment_completed", count: 1, timestamp: ... }
```

No personal identifiers.

2.2 Engine KPIs (Critical)

Track intelligence engine performance:

Matching Engine

- match duration
- match failures
- distribution of match scores
- top match reasons (anonymous)

Assessment Engine

- completion duration
- error rate
- average capability distributions

Learning Engine

- recommendation latency

- module uptake rate

Projection Engine

- projection duration
- readiness distribution

None of these metrics include names, emails, or sensitive attributes.

2.3 System Health Metrics

- SSR page render time
- server action latency
- DB query latency
- error rate
- RLS denials
- uptime
- Sentry error frequency

2.4 Business-Level KPIs

Anonymous aggregated performance indicators:

- number of active Talents
- number of active Houses
- number of opportunities created
- number of introductions requested
- number of introductions accepted

- conversion from match → interaction

Again **anonymous**, and **never tied to real people**.

2.5 Search & Matching Metrics

Privacy-safe insights like:

- distribution of division filters
- role levels most requested
- store tiers most matching
- geographic search patterns

All aggregated and anonymized.

---

3. What Tailor Shift NEVER Tracks (Strictly Forbidden)

To remain luxury-appropriate and compliant:

❌ No user-level tracking

No persistent user behaviour analytics.

❌ No personalised behaviour logs

No heatmaps, no scroll tracking, no session replay.

❌ No tracking of rejection reasons

Declines and acceptances remain private; no analysis of “reasons”.

❌ No cookie-based profiling

Only essential cookies (Auth.js) allowed.

❌ No cross-site tracking IDs

No Facebook Pixel, no Google Ads remarketing.

❌ No fingerprinting

Browser or device fingerprinting forbidden.

❌ No sensitive analytics

No assessment answers.
No compensation.
No preferences.
No personal traits.

❌ No individual-level export for analysis

Only aggregated metrics allowed for product improvement.

---

4. Data Pipeline Architecture

Tailor Shift analytics follow a minimal pipeline:

```
Frontend events (anonymized)
       ↓
Vercel Analytics or internal collector
       ↓
Aggregate storage (no PII)
       ↓
Dashboards (anonymous)
```

No direct database involvement for analytics.
No user-level identifiers stored.

---

5. Event Structure (V5 Canon)

All analytics events follow the same shape:

```
{
  event: "string",
  context: {
    segment: "public|talent|brand",
    engine_version: "MatchingEngine_v1_2026_Q1",
    page: "/talent/dashboard",
    device: "mobile|desktop"
  },
  metadata: {
    duration_ms?: number,
    status?: "ok" | "error",
    score_bucket?: "0-20" | "21-40" | ...
  },
  timestamp: ISO8601
}
```

Forbidden fields:

- user_id
- email
- talent_id
- brand_id
- names
- location

---

6. Key Metrics & KPIs (Luxury Context)

Tailor Shift improves around **value**, **efficiency**, and **precision**.

6.1 Talent KPIs

- profile completion rate

- assessment completion rate
- average capability distribution
- match quality distribution
- opportunity view-to-interest ratio
- interest-to-introduction acceptance ratio

6.2 Brand KPIs

- opportunity fill activity
- match quality distribution
- introduction acceptance rate
- time-to-first-match
- search result refinement patterns

6.3 Marketplace KPIs

- volume of introductions
- brand engagement volume
- talent engagement volume
- segment-level activity heatmap (anonymous)

6.4 Engine KPIs

- matching accuracy signals (anonymous)

- matching response times
- assessment scoring latency
- projection recommendation latency

These help continuously refine the engines.

---

7. Dashboards (Internal Only)

Dashboards are **internal**, used by:

- Engineering
- Product
- QA
- Founders

No external analytics dashboards visible to Houses or Talents.

Dashboards include:

- engine performance
- RLS denials
- server action latency
- SSR latency
- errors over time
- matching score distributions

---

8. Data Retention Rules (Forward-declared)

8.1 Analytics Data

- retention: 90 days (non-sensitive)
- aggregated summaries kept 12 months

8.2 Logs

- error logs: 30–60 days
- anonymised logs: long-term
- sensitive logs: none

Links directly with Chapter 29.

---

9. Compliance & GDPR Considerations

Analytics must always be:

- anonymized
- aggregated
- PII-free
- respectful of consent
- non-inferential
- compliant with EU AI Act low-risk classification

No profiling.
No sensitive information.
No tracking cookies.

---

10. Testing Analytics (QA Requirements)

QA must verify:

- no PII in analytics events
- correct event firing
- correct anonymous context
- no sensitive fields in payload
- no analytics calls in client components for private flows

CHAPTER 29 — BACKUP, DATA RETENTION & DATA DELETION (V5 CANON)

Data safety, deletion correctness, and long-term integrity for a luxury intelligence platform.

---

0. Purpose — Why Backups & Retention Are Critical

Tailor Shift stores sensitive professional data:

- Talent identity fields
- Experience, assessment summaries, preferences
- Brand store structures
- Opportunities
- Interactions

Therefore we need:

- guaranteed recovery

- controlled retention
- lawful deletion
- GDPR compliance
- structured archival
- no accidental data loss

This chapter establishes **the canonical rules** that allow Tailor Shift to be fully recoverable, compliant, and trustworthy.

---

1. Backup Policy (Tiger Cloud PostgreSQL)

Backups are managed by Tiger Cloud and integrated with Tailor Shift DevOps.

1.1 Automated Backups

- hourly backups (rolling 24–48h)
- daily snapshots (rolling 7–14 days)
- weekly backups (rolling 4–8 weeks)

1.2 Backup Storage

- encrypted at rest
- stored in multi-region geo-redundant storage
- isolated per environment

1.3 Backup Integrity Tests

Monthly:

1. restore backup into staging

2. run smoke tests:
    - RLS checks
    - onboarding flows
    - opportunity creation
    - matching engine
3. validate schema consistency
4. sign-off required

---

2. Recovery Procedure (Disaster-Class Only)

In case of major outage / data corruption:

Step 1 — Activate Recovery Mode

- freeze production write operations
- display maintenance banner (clean UI)

Step 2 — Deploy Restored Backup to Staging

- validate engine logic
- validate stores, opportunities, interactions
- validate indexing

Step 3 — Promote Restored Backup to Production

- minimal downtime
- post-validation checks

- clear logs
- re-enable write operations

Step 4 — Incident Report

Generated internally (Ch. 30).

---

3. Data Retention Policy (Per Data Type)

Tailor Shift operates under a strict retention policy to respect:

- GDPR
- industry norms
- luxury privacy expectations
- minimalism philosophy

3.1 Talent Data

Data	Retention
identity (first/last name, email)	until deletion request
profile fields	until deletion request
experience blocks	until deletion request
preferences	until deletion request
capability summary	until deletion request
compensation profile	until deletion request
assessment answers	**never retained long-term** → deleted immediately after scoring

---

3.2 Brand Data

Data	Retention
brand identity	persistent
stores	persistent
opportunities	persistent while active
interactions	persistent until anonymization policy applies

3.3 Derived Data

Data	Retention
match snapshots	90 days max
cached engine results	7–30 days
analytics (aggregate)	up to 12 months
anonymized statistical data	unlimited

3.4 Logs & Observability

Data	Retention
server action logs	30–60 days
error logs	30 days
anonymized metrics	unlimited
Sentry traces	7–14 days

---

4. Data Deletion — Full Lifecycle

Deletion must be:

- **instant**
- **complete**
- **irreversible**
- **GDPR-compliant**
- **auditable**

Tailor Shift implements a **two-phase deletion**.

4.1 Phase 1 — User-Initiated Deletion Request

Triggered by:

- Talent
- Brand Admin
- Admin on behalf of user (GDPR request)

UI copy:

> “Your account and associated data will be permanently deleted.”

System actions:

1. validate identity
2. mark user as “pending deletion”
3. schedule asynchronous deletion job

4.2 Phase 2 — Hard Deletion

Performed by server process:

Step A — Delete `auth.users` row

This cascades:

- sessions
- emails
- OAuth links
- device identifiers

Step B — Cascade-delete domain data

Through DB foreign keys:

- talents row
- experience_blocks
- assessments
- learning_progress

Step C — Anonymize Interactions

Replace Talent ID with:

```
"talent_anonymized_" + uuid
```

Interaction structural integrity preserved; PII erased.

Step D — Purge Sensitive Fields

Remove:

- compensation_profile
- career_preferences
- assessment_summary

Step E — Clean Logs

Any logs referencing the user indirectly must be scrubbed (only metadata exists anyway).

Step F — Verification

System confirms:

- no remaining references
- RLS denies access (double-check)
- analytics unaffected (aggregate only)

---

5. Brand-Initiated Deletion

If a Brand owner deletes the Brand:

1. delete `brands` row
2. cascade-delete stores
3. cascade-delete opportunities
4. anonymize interactions
5. leave Talents untouched (they own their profiles)

---

6. Special Case: Assessment Answer Deletion

Assessment answers (raw data) are:

- stored only for scoring
- immediately discarded
- never shown to Brand
- never used for analytics
- never stored long-term

This keeps Tailor Shift compliant by design.

---

7. Sensitive Data Scrubbing

7.1 Hourly Job

Scrubs:

- orphaned logs
- expired match snapshots
- abandoned interactions (over 6 months old)

7.2 Weekly Job

Scrubs or compresses:

- old analytics
- old monitoring traces

7.3 Monthly Job

Reviews:

- storage footprint
- retention compliance

---

8. Right to Access (GDPR Article 15)

Tailor Shift must provide downloadable export of:

- Talent profile
- experience blocks

- preferences
- capability summaries
- learning progress
- interactions involving them

Format:

```
JSON file
export_YYYYMMDD.json
```

Compiled server-side through server actions.

---

9. Right to Rectification (GDPR Article 16)

User can edit:

- name
- experience
- preferences
- assessment summary (via new assessment, not manual edits)
- mobility
- email

Contact support required for email or identity updates.

---

10. Right to Restrict Processing (GDPR Article 18)

Talent can temporarily pause:

- matching
- opportunity recommendations
- brand visibility

UI option:

> “Pause my profile visibility”

Effects:

- remove from brand search
- freeze matching
- keep profile internally accessible

---

11. Right to Data Portability (GDPR Article 20)

Exports are:

- machine-readable (JSONL)
- semantically structured
- aligned with canonical entities

---

12. Luxury-Specific Privacy Requirements

Tailor Shift adheres to elevated privacy standards expected by luxury houses:

- all Talent data private by default

CHAPTER 30 — INCIDENT MANAGEMENT & PLAYBOOKS (V5 CANON)

Structured, calm, predictable response to operational issues. Zero chaos. Total control.

---

0. Purpose — Why Incident Management Matters

In luxury retail, reliability equals credibility.
Tailor Shift must demonstrate:

- excellence under pressure,
- absolute data security,
- calm communication,
- rigorous procedures,
- predictable recovery paths.

This chapter establishes:

- how incidents are detected,
- how they're classified,
- how they're responded to,
- how the team communicates,
- how the platform returns to safety,
- how incidents are documented and learned from.

---

1. What Counts as an Incident (Canonical Definition)

An **incident** is any event that:

- degrades the platform,
- prevents expected behaviour,
- risks data integrity,
- impacts privacy or security,
- affects user trust.

Incidents fall into four classes.

---

2. Incident Classification (V5)

2.1 Severity 1 — Critical (“Outage / Data Risk”)

Examples:

- Production database unavailable
- RLS misconfiguration causing data leakage
- authentication outage
- introduction acceptance not working globally
- engines failing completely
- deployment rollback required
- corrupted schema

Severity 1 incidents require immediate action.

2.2 Severity 2 — High (“Major Function Degraded”)

Examples:

- brand cannot create opportunities
- talent cannot update profile
- assessment failing for some users
- matching engine returning errors
- server actions consistently failing
- Sentry error spike

Requires rapid triage and fix.

2.3 Severity 3 — Medium (“Partial Degradation”)

Examples:

- some stores not saving
- intermittent errors
- performance regression
- non-critical routes misbehaving
- minor UI breakage on mobile

Requires scheduled fix.

2.4 Severity 4 — Low (“Non-Blocking Defect”)

Examples:

- wording issues

- design inconsistencies
- minor component misalignment
- analytics event not firing

Handled in next release.

---

3. Detection & Monitoring

Incidents are detected through:

3.1 Automated Systems

- Sentry alerts
- Vercel deployment status
- API error spikes
- engine failure logs
- DB slow-query alerts
- RLS denial spikes
- health checks

3.2 Manual Signals

- QA detection
- founder review
- internal testing
- user report (rare)

---

4. Incident Response Workflow

A canonical 7-step workflow ensures consistency.

Step 1 — Detection

Signal captured via Sentry, logs, or monitoring system.

Step 2 — Triage

Responsible engineer classifies severity:

- S1 → immediate
- S2 → rapid
- S3 → scheduled
- S4 → backlog

If security issue → classify as **S1** automatically.

Step 3 — Assignment

The incident is assigned to:

- Engineering (backend, frontend, or infra)
- optionally Product if UX-level
- Admin for compliance-related events

Step 4 — Containment

Goal: stop the issue from spreading.

Examples:

- disable specific server actions
- temporarily disable opportunity creation
- remove access to brand interactions
- freeze new signups
- shift traffic from edge to core server

Containment must be:

- safe
- reversible
- minimal

Step 5 — Remediation

Fix is implemented following V5 guidelines:

- no patching directly in production
- create a `hotfix/*` branch
- deploy to staging
- confirm via smoke tests
- promote to production
- validate results via monitoring

Step 6 — Post-Validation

Verify:

- no further regressions
- matching engine stable
- profile updates functional
- no residual RLS issues
- performance restored

Step 7 — Documentation

An incident report is required (internal-only):

```
Incident ID
Date
Severity
Summary
Root Cause
Impact
Fix Applied
Lessons Learned
Prevention Actions
```

---

5. Playbooks (Canonical Response Guides)

Below are Tailor Shift’s official playbooks.

5.1 Playbook: Production Outage (S1)

1. Notify team (Ops channel)
2. Freeze deployments

3. Activate maintenance banner
4. Verify DB connectivity
5. Check logs for engine loops
6. Check RLS rules (common break source)
7. Roll back to last stable build if needed
8. Validate recovery using smoke tests
9. Remove maintenance banner
10. Document incident

5.2 Playbook: RLS Breach / Data Exposure (S1)

Absolutely critical.

1. Immediately restrict DB access
2. Disable all mutations via feature flag
3. Terminate all active sessions
4. Restore DB from last known-good backup
5. Reapply migrations
6. Validate RLS with automated suite
7. Redeploy stable build
8. Full incident documentation
9. Prepare compliance report (Ch. 31)

5.3 Playbook: Authentication Failure (S1–S2)

1. Validate Auth.js provider status
2. Validate environment variables
3. Check callback URLs
4. Validate session cookie encryption
5. Restart edge/Auth process
6. Fix environment keys if needed
7. Re-test login, signup, reset

5.4 Playbook: Matching Engine Degradation (S2)

1. Check engine logs
2. Validate engine version
3. Validate underlying opportunity/talent data
4. Check for malformed input
5. Re-run engine tests locally
6. Patch engine logic
7. Deploy hotfix
8. Validate scoring consistency

5.5 Playbook: Assessment Engine Failing (S2)

1. Inspect logs for malformed answer sets
2. Validate scoring pipeline
3. Test engine with canonical scenarios

4. Fix code or question mapping
5. Redeploy
6. Validate dashboard integration

5.6 Playbook: Interaction Engine Failure (S2)

1. Inspect interaction table
2. Validate uniqueness constraint
3. Validate foreign keys
4. Patch server action
5. Re-test acceptance on staging
6. Promote to production

5.7 Playbook: Slow Performance (S3)

1. Identify slow routes
2. Inspect DB slow query log
3. Add missing indexes
4. Optimize SSR queries
5. Cache engine results
6. Validate improvements

5.8 Playbook: Frontend Rendering Errors (S3)

1. Check Sentry error
2. Reproduce locally
3. Patch component
4. Add regression test
5. Deploy

---

6. Communication Rules (Internal Only)

6.1 Tone:

- calm
- factual
- concise
- non-emotional
- zero speculation

6.2 Who Is Informed

- Engineering
- Product
- Founders

6.3 What Is Not Shared

- specific user data
- sensitive logs
- system-level secrets

7. Post-Incident Review (PIR)

Within 72 hours:

- root cause analysis
- timeline reconstruction
- regression test additions
- RLS policy review
- engine version review
- commit to prevention actions

No blame.
Only structural fixes.

8. Preventive Measures

Mandatory for all production systems:

- RLS tests before releases
- engine validation suite
- no untested migrations
- strict CI/CD gating
- drift detection (Terraform)
- hourly snapshots
- daily performance reports

CHAPTER 31 — LEGAL, COMPLIANCE & DATA GOVERNANCE (V5 CANON)

The complete legal & compliance doctrine of Tailor Shift. Privacy as a product, trust as infrastructure.

---

0. Purpose — Why a Governance Chapter Exists

Luxury houses expect:

- world-class privacy
- zero compromise
- full explainability
- mature governance
- compliance with GDPR & EU AI Act
- clarity on dataflow
- predictable legal posture

Tailor Shift is built to satisfy these expectations by design.

---

1. Governance Philosophy — “Privacy as a First-Class Feature”

Tailor Shift integrates legal and ethical responsibility into its architecture:

- no unnecessary data
- no behavioural tracking

- no opaque machine-learning models
- deterministic explainable engines
- no exposure of sensitive fields
- data minimisation everywhere
- user sovereignty
- auditable systems
- reversible actions
- compliant defaults

Privacy is not a checkbox — it is a core product value.

---

2. Legal Foundations

Tailor Shift complies with:

- **GDPR**
- **EU AI Act (low-risk category)**
- **Digital Services Act**
- **Consumer rights legislation (EU/US)**
- **Global privacy standards (where applicable)**
- **ISO-inspired security practices** (not certified, but aligned)

Tailor Shift does **not** process:

- biometric data
- sensitive personal data
- protected category data

- psychometric data
- behavioural profiling data

Tailor Shift operates solely on:

- professional information
- preference data (non-sensitive)
- capability summaries
- career signals
- structured professional experience

---

3. GDPR Compliance Model

Compliance mapped across the full lifecycle.

3.1 Lawful Basis for Processing

Under GDPR Article 6:

- **contractual necessity**: operating Tailor Shift service
- **user consent**: specific optional features
- **legitimate interest**: maintaining platform security
- **compliance obligation**: legal requirements

3.2 Data Controller / Processor Roles

- **Tailor Shift** → Data Controller (Talent & Brand user data)

- **Service providers** → Data Processors (hosting, logging)

No sub-processors handling sensitive personal data.

3.3 Data Subject Rights

Implemented fully:

- right of access
- right to rectification
- right to erasure
- right to restriction
- right to data portability
- right to object
- right to withdraw consent

All rights implemented via:

- in-app flows
- or via support request (verified identity)

3.4 Consent Management

Consent is required for:

- receiving email notifications
- enabling personalised recommendations
- participating in introductions
- optional profiling (if ever added in V6)

Consent is:

- explicit
- recorded
- reversible
- unobtrusive

3.5 Minimisation & Purpose Limitation

Tailor Shift stores **only** what is strictly required:

- no assessment answers
- no compensation details surfaced
- no sensitive categories
- no raw behavioural signals
- no telemetry PII

Purpose limitation enforced via architecture:

- RLS
- engine-level abstraction
- anonymized analytics
- strict server-only operations

---

4. EU AI Act Compliance

Tailor Shift is explicitly designed to fall under the **"Low-Risk AI System"** category.

4.1 Reasons Tailor Shift Is Low-Risk

- uses **deterministic engines** (no ML inference)
- offers **explainable outputs**
- does **not score humans psychologically**
- only uses **professional capability signals**
- does **not** assign automated hiring decisions
- always requires **human review**
- does **not** profile race, gender, age, or sensitive categories

4.2 Mandatory Transparency

Tailor Shift:

- shows how match scores are computed
- displays the breakdown
- uses neutral, factual language
- uses stable weighting
- version stamps engines

4.3 Human Oversight

Brands:

- cannot see private Talent data
- cannot auto-reject based on score
- must request human-reviewed introductions

Talents:

- always approve contact sharing
- control their visibility

---

5. Data Governance Framework (Canonical)

Tailor Shift operates a **four-pillar data governance model**.

5.1 Pillar 1 — Data Inventory & Classification

Data Types

- **Public**: storefront pages
- **Internal**: anonymized analytics
- **Confidential**: Talent profile
- **Restricted**: compensation profile
- **Critical**: authentication & RLS policies

No sensitive special-category data is collected.

5.2 Pillar 2 — Data Flow Documentation

Canonical flows:

For Talents:

```
Talent → Profile → Engines → Dashboard
      → Interaction → Consent → Brand
```

For Brands:

```
Brand → Stores → Opportunities → Engines → Dashboard
```

For System:

```
Assessments → Scores → Engines → Summaries
(No raw answers stored)
```

Flows documented and reviewed quarterly.

5.3 Pillar 3 — Data Access Controls

Enforced with:

- Auth.js
- RBAC
- Next.js middleware
- RLS
- server-side rendering
- anonymization routines
- structured server actions

No client-side data access to protected resources.

5.4 Pillar 4 — Data Lifecycle Management

Outlined in Chapter 29:

- backups
- retention
- anonymization
- deletion
- restoration

Governance ensures:

- reversibility
- compliance

- minimal risk

---

6. Third-Party Services & Contracts

Tailor Shift may rely on:

- Vercel (deployment)
- Tiger Cloud (database)
- Sentry (error monitoring)
- Email provider (transactional)

All third parties must:

- sign data processing agreements (DPA)
- comply with GDPR
- provide encryption
- have no access to user data except metadata
- hold ISO-level certifications

No third party receives:

- assessment summaries
- compensation
- interaction data
- raw Talent or Brand profiles

---

7. Cookies & Tracking

Allowed (Strictly Necessary)

- Auth.js session cookie
- CSRF tokens
- essential runtime cookies

Forbidden

- tracking cookies
- analytics cookies requiring consent
- fingerprinting
- retargeting cookies
- marketing pixels

Tailor Shift's analytics layer (Ch. 28) is fully anonymized and cookie-free.

---

8. Data Breach Policy (Summary)

If a breach is detected:

1. classify severity
2. freeze system operations
3. secure database access
4. investigate via logs & snapshots
5. restore from backup if needed
6. notify affected users (legal requirement)
7. notify authorities (if required)
8. document in incident report (Ch. 30)

Breach responses follow GDPR Articles 33–34.

---

9. Contractual & Legal Documentation

Tailor Shift must maintain:

- Terms of Service
- Privacy Policy
- Data Processing Agreement (DPA)
- Incident Response Policy
- Data Retention Document
- RLS Policy Sheet
- Engine Explainability Note
- AI Transparency Statement (EU AI Act)

All documents are versioned with each major release.

---

10. Regulatory Alignment Checklist

Before going live or signing with luxury houses, Tailor Shift must satisfy:

- GDPR-compliant privacy policy
- User-level data deletion implemented
- Right to access export flow implemented
- No raw assessment answers stored
- No compensation leaked
- RLS enabled on all domain tables

- Engines versioned + documented
- Cookie policy minimal & compliant
- No invasive analytics
- Incident playbooks ready
- Backup & restore validated
- Penetration tests passed

CHAPTER 32 — APPENDICES & GLOSSARY (V5 CANON)

The complete reference atlas of Tailor Shift. Taxonomies, matrices, weights, terms, definitions.

SECTION A — MASTER ENUMERATIONS

These enumerations must match the PostgreSQL `ENUM` definitions defined in Ch.17.

A.1 DIVISIONS (division_enum)

```
fashion
leather_goods
shoes
beauty
fragrance
watches
high_jewelry
eyewear
accessories
```

A.2 REGIONS (region_enum)

```
emea
americas
apac
middle_east
```

A.3 STORE TYPES (store_type_enum)

```
flagship
main_boutique
department_corner
travel_retail
outlet
pop_up
franchise
partner
hJ_salon
```

A.4 COMPLEXITY TIERS (complexity_tier_enum)

```
T1 (XXL flagship)
T2 (major boutique)
T3 (standard boutique)
T4 (resort/seasonal)
T5 (outlet)
```

A.5 OPENNESS STATUS (openness_enum)

```
actively_looking
open_to_opportunities
not_looking
```

SECTION B — ROLE LADDER (L1–L8)

The canonical Levels for all luxury retail roles:

Level	Canonical Title	Essence
L1	Sales Advisor	execute service rituals
L2	Senior Advisor	elevated FOH, clienteling signals
L3	Floor/Department Manager	team coordination
L4	Assistant Store Manager	N-1 operational leadership
L5	Store Manager	full boutique ownership
L6	Flagship Director	multi-division governance
L7	Cluster/Area Manager	multi-boutique leadership
L8	Retail/Regional Director	market-level governance

All real-world titles map to these levels.

SECTION C — EXPERIENCE BLOCK TYPES

```
FOH
BOH
Leadership
Clienteling
Operations
Business
```

Blocks are atomic units of real work.

SECTION D — CAPABILITY MODEL (4 Dimensions)

Used in the Assessment Engine V5.

```
Service Excellence
Clienteling & Relationship Building
Operational Reliability
Leadership Signals
```

Future V6 adds the full 7D model.

SECTION E — MATCHING ENGINE (V5 CANONICAL WEIGHTS)

E.1 Weighting Breakdown

Dimension	Weight
Role Fit	20%
Division Fit	20%
Store Context Fit	15%
Capability Fit	15%
Geography Fit	10%
Experience Block Fit	10%
Preference Alignment	10%

Total = 100%

E.2 Role Fit Map

Δ Role Level	Score
0	100
1	85

2	55
3	25
≥4	0

E.3 Division Fit Map

```
intersection_ratio = |intersection| / |required|
scores:
1.0 → 100
0.66 → 80
0.33 → 50
0.0 → 0
```

E.4 Store Tier Fit

Tier Difference	Score
0	100
1	80
2	55
3	20
≥4	0

E.5 Compensation Fit (abstract-only)

Relative Diff	Score
≤ 25%	100
≤ 50%	80
≤ 100%	60
> 100%	40

UX tag only; raw values never shown.

SECTION F — ASSESSMENT ENGINE MAP (V5)

F.1 4D Model

- service
- clienteling
- operations
- leadership

F.2 Output

```
{
  service_excellence: number,
  clienteling: number,
  operations: number,
  leadership_signals: number
}
```

F.3 Insight Categories

- “Strong signal” >75
- “Growing” 60–75
- “Developing” 40–60
- “Needs attention” <40

SECTION G — LEARNING ENGINE PRIORITY MATRIX

G.1 Gap Severity → Priority

Gap Score	Priority
<40	High
40–60	Medium
>60	Low

G.2 Ranking Logic (V5)

```
score =
  0.45 * gap_severity +
  0.30 * trajectory_relevance +
  0.20 * division_context +
  0.05 * novelty_factor
```

SECTION H — PROJECTION ENGINE CANON

H.1 Typical Role Timelines

Transition	Range
L1 → L2	6–12 months
L2 → L3	9–18 months
L3 → L4	12–24 months
L4 → L5	18–30 months

L5 → L6	24–42 months

H.2 Projection Output

```
{
  next_role_level: number,
  next_store_tier: tier,
  projected_hubs: string[],
  role_timeline_range: [min, max],
  tier_readiness: string,
  capability_gaps: string[],
  recommended_focus: string[],
  division_trajectory: string[],
  notes: string
}
```

SECTION I — INTERACTION ENGINE STATE MACHINE

The canonical 4-state lifecycle:

```
requested  → accepted
sent       → declined
```

Where:

- `requested` = Talent origin
- `sent` = Brand origin
- `accepted` = mutual consent
- `declined` = final refusal (irreversible)

Contact unlocks only at `accepted`.

SECTION J — ERROR CODES (Canonical)

```
AUTH_ERROR
FORBIDDEN
VALIDATION_ERROR
INPUT_FORMAT_ERROR
ENGINE_ERROR
DATABASE_ERROR
UNKNOWN_ERROR
```

All server actions and engines must return one of these.

SECTION K — PRIVACY DIAGRAMS

K.1 Talent → Brand Visibility

Before Acceptance:

- division experience
- role level
- years
- assessment summary
- location
- experience blocks (summary)

Never:

- compensation
- preferences
- raw assessment answers

- projections
- learning history
- private notes

After Acceptance:

- contact details (minimal only: email, phone if given)

SECTION L — APP ROUTER SITEMAP (Canonical)

Condensed full tree:

```
/ (public)
  /professionals
  /brands
  /about
  /resources
  /login
  /signup
  /terms
  /privacy

/talent
  /dashboard
  /profile
  /assessment
  /learning
  /opportunities
  /settings

/brand
  /dashboard
  /profile
  /stores
  /opportunities
  /search
  /talent/[id]
  /settings

/api (webhooks only)
```

No other routes allowed unless added in future versions.

SECTION M — TESTING MATRIX (Complete)

Unit tests:

- matching
- assessment
- learning
- projection

Integration tests:

- onboarding
- opportunity creation
- interaction flow
- RLS boundaries

Performance tests:

- SSR < 200 ms p50
- matching < 250 ms
- assessment < 200 ms

Security tests:

- SQL injection
- RLS leaks
- session hijack
- OAuth misrouting

SECTION N — GLOSSARY (V5 Canon)

Brand / House

A luxury retail organization using Tailor Shift to manage opportunities.

Talent

A retail professional creating a profile on Tailor Shift.

Role Level (L1–L8)

Canonical ladder describing professional seniority.

Complexity Tier (T1–T5)

Operational complexity classification of store environments.

Experience Block

Structured representation of a specific area of professional practice.

Capability Summary

Aggregated score of service, clienteling, operations, leadership.

Matching Engine

Deterministic scoring mechanism between Talent and Opportunity.

Assessment Engine

Mechanism generating capability summary via structured questionnaire.

Learning Engine

Recommends modules to strengthen capability gaps.

Projection Engine

Suggests plausible next roles, tiers, and career paths.

Interaction Engine

Consent-based handshake system between Talent and House.

RLS (Row-Level Security)

Database-level access boundary ensuring zero data leakage.

Server Action

Server-side function used for safe mutations.

SSR (Server-Side Rendering)

Rendering pages on the server to ensure correct data + privacy.

Zero-Trust

Security philosophy assuming no implicit trust at any layer.