DPA Guard — Technical Design Document (TDD)

1. Document Control

- **Product:** DPA Guard (MVP)
- **Doc Type:** Technical Design Document (TDD)
- **Version:** 1.0 (Draft)
- **Status:** In Review
- **Owner:** Engineering
- **Audience:** Engineering, Security, (Advisory) Legal/Privacy
- **Last Updated:** 2026-01-10

---

2. Overview

DPA Guard is a web application that ingests a **DPA (PDF/DOCX)**, extracts and segments its text, classifies clauses into standardized GDPR/DPA topics, evaluates each topic against a **versioned Playbook**, and produces: - Executive summary (decision + top risks) - Clause-by-clause risk table with **evidence quotes** - Negotiation pack (commentary table + email draft) - PDF export

Core design principles: - **Evidence-first:** Every flagged item must be backed by exact quotes from the document. - **Low hallucination:** LLM outputs are constrained; evidence is validated deterministically. - **Playbook-driven:** Judgments come from rules, not "general AI opinion." - **EU trust baseline:** secure storage, transparent retention, tenant isolation.

---

3. Architecture Summary

3.1 High-level Components

- **Web Frontend:** Next.js (upload, context, results, export)
- **API Backend:** FastAPI (auth, review lifecycle, orchestration)
- **Worker:** Celery (or RQ) + Redis queue for async processing
- **Database:** Postgres (users/orgs/reviews/segments/results/audit)
- **Object Storage:** S3-compatible bucket (uploaded docs + generated PDFs)
- **LLM Provider:** Hosted LLM (prefer EU region); used for classification, evaluation, and optional verification
- **Vector Store (optional MVP):** pgvector (playbook retrieval only)

3.2 Data Flow (Lifecycle)

1) User creates review + optional context 2) Upload DPA → stored in object storage 3) Worker pipeline runs: - extract → segment → classify → retrieve playbook → evaluate → validate evidence → compose outputs 4)

Results stored in DB + PDF stored in object storage 5) UI renders summary + table; user exports negotiation pack/PDF

3.3 Diagram (Text)

```
Browser (Next.js)
  |  HTTPS
  v
FastAPI (API)
  |  enqueue job
  v
Redis Queue  ---> Worker (Celery/RQ)
                    |
                    | \--> Postgres (reviews, segments, results)
                    |
                    |  \-> Object Storage (uploads, exports)
                    |
                    \--> LLM Provider (classification/evaluation/verifier)
                    \--> Playbook Store (DB + optional pgvector)
```

---

4. Core Data Model (Proposed)

4.1 Entities

organizations - id, name, created_at

users - id, org_id, email, name, role, created_at

reviews - id, org_id, created_by - vendor_name (nullable) - doc_filename, doc_mime, doc_storage_key - playbook_version - context_json (role, region, sensitivity, etc.) - status: CREATED | UPLOADED | PROCESSING | COMPLETED | FAILED - created_at, updated_at

review_segments - id, review_id - segment_index - heading (nullable), section_number (nullable) - page_start/page_end (nullable) - text - hash (for dedupe/debug)

segment_classifications - id, review_id, segment_id - clause_type - confidence - method: RULES | LLM | HYBRID

clause_evaluations (one per clause_type per review) - id, review_id - clause_type - risk_label: GREEN | YELLOW | RED - short_reason - suggested_change - triggered_rule_ids (array) - evidence_spans (json array) ← **source of truth**

evidence_spans (stored inline as JSON, or separate table if needed) - segment_id - start_char, end_char - quote_text (derived; must match substring) - note (optional)

review_outputs - review_id - exec_summary_json - negotiation_email_draft - pdf_storage_key

audit_log - id, org_id, user_id, review_id - action: UPLOAD | VIEW | EXPORT | DELETE - timestamp, metadata_json

4.2 Clause Type Enum (MVP)

Use a fixed enum aligned to PRD clause coverage (15 types). This ensures table consistency and predictable playbook retrieval.

---

5. Processing Pipeline (MVP)

5.1 Step A — Ingestion & Text Extraction

Goal: extract structured text with section cues.

- PDF (text-based): use a parser that preserves layout as much as feasible (headings, page numbers).
- DOCX: use python-docx style extraction.

Outputs: - raw_text - per-page text (if available) - extraction metadata (page count, text density)

Scanned PDF detection: - If text density below threshold → mark as UNSUPPORTED_OCR and fail gracefully with guidance.

5.2 Step B — Segmentation

Goal: split into logical segments for classification/evaluation.

Approach (hybrid): - Rule-based segmentation using: - numbering patterns (1., 1.1, (a), etc.) - headings capitalization and whitespace cues - page boundaries - Optional LLM assist for difficult docs (limited to restructuring, not judging).

Outputs: review_segments[]

5.3 Step C — Clause Classification

Goal: map segments → clause types.

Approach: 1) **Rules-first classifier** (keywords + patterns) 2) If low confidence or ambiguous, call LLM classifier with strict schema output

Outputs: segment_classifications[]

5.4 Step D — Playbook Retrieval (Light RAG)

Goal: provide only relevant rules to the evaluator.

Approach: - Primary key retrieval: clause_type → playbook rules - Optional similarity search for sub-variants (pgvector), constrained to playbook content only

Outputs: playbook_snippet (rules list for clause_type)

5.5 Step E — Evaluation (Playbook-based)

Goal: compute risk per clause_type using clause text + context + playbook.

Inputs: - concatenated relevant segment text(s) - context_json - playbook rules

LLM output (strict JSON schema): - risk_label - short_reason (<= 3 sentences) - suggested_change (negotiation ask; fallback wording only if present in playbook) - candidate evidence quotes (text-only candidates) - triggered_rule_ids

5.6 Step F — Evidence Extraction & Validation (Hard Guardrail)

Goal: ensure every quote is real and traceable.

Process: 1) For each candidate quote, locate it within the original segment text. 2) If found → store span (segment_id, start_char, end_char). 3) If not found → reject that quote and either: - re-prompt evaluator to provide matching quotes, or - downgrade output to “missing/unclear” with explicit note

Rule: The system must only display evidence that passes deterministic substring/span validation.

5.7 Step G — Optional LLM Verifier (Secondary Guardrail)

Goal: catch reasoning mismatches even when quotes exist.

Verifier checks: - Do the validated quotes support the stated reason? - Are any claims made that lack support?

Verifier cannot introduce new evidence; it can only: - approve - request re-evaluation - flag as “needs human review”

5.8 Step H — Output Composition

Generate: - exec summary JSON - clause table rows - negotiation pack (table + email draft) - PDF export

---

6. Prompting Strategy & Schemas

6.1 General

- Use low temperature for determinism.
- Use JSON-schema-constrained outputs.
- Provide only: clause text + context + playbook snippet.

6.2 Classification Prompt (Schema)

Output: - segment_id - clause_types[] - confidence - rationale (short)

6.3 Evaluation Prompt (Schema)

Output: - clause_type - risk_label - short_reason - suggested_change - triggered_rule_ids[] - candidate_quotes[] (short exact excerpts)

6.4 Verifier Prompt (Schema)

Output: - status: PASS | FAIL | REVIEW - issues[]

---

7. API Design (MVP)

7.1 Authentication

- MVP options:
- Email magic link (simple)
- Or OAuth (Google/Microsoft) if needed
- All API requests scoped to org.

7.2 Endpoints (Proposed)

- `POST /reviews` → create review (context, vendor_name)
- `POST /reviews/{id}/upload` → upload file (multipart)
- `POST /reviews/{id}/start` → enqueue processing job
- `GET /reviews/{id}` → review metadata + status
- `GET /reviews/{id}/results` → exec summary + table
- `GET /reviews/{id}/export/pdf` → signed URL or streamed PDF
- `GET /playbook/versions` → list versions

7.3 Job Events

- Worker writes status updates to DB
- Optional: WebSocket/SSE for live progress

---

8. UI Rendering Requirements (Technical)

8.1 Evidence Display

- Show quote with section/heading reference
- Provide “jump to segment” (modal showing full segment text)

8.2 Copyable Outputs

- Export table as clipboard-friendly TSV/Markdown
- Provide vendor email draft with one-click copy

---

9. PDF Generation

9.1 Content

- Page 1: executive summary + top risks
- Subsequent pages: clause table
- Optional appendix: negotiation email + commentary table

9.2 Implementation

- Server-side PDF generation (Python) using a stable PDF library.
- Store generated PDF in object storage and return via signed URL.

---

10. Security, Privacy, and Compliance

10.1 Tenant Isolation

- Every entity keyed by org_id; enforce at API and DB query layer.

10.2 Storage & Encryption

- TLS in transit
- Encrypt at rest (object storage + DB)

10.3 Retention & Deletion

- Define retention policy (configurable)
- Support delete-on-demand:
- delete objects (original + exports)
- delete DB rows (or soft-delete with purge job)

10.4 Logging

- Never log raw contract text in application logs.
- Store text in DB with access controls; redact in logs.

10.5 “Not Legal Advice” Controls

- UI + PDF includes disclaimers

---

11. Observability & Operations

11.1 Metrics

- processing time per step
- LLM call counts, latency, error rate
- % documents failing extraction/segmentation

11.2 Tracing

- Correlate pipeline steps by review_id
- Store intermediate artifacts (segments, classifications) for debugging

11.3 Alerting

- Worker failures
- abnormal latency spikes
- storage access errors

---

12. Testing Strategy

12.1 Golden Test Set

- ≥20 representative DPAs
- Evaluate:
- clause coverage
- evidence validity
- stability across playbook versions

12.2 Unit Tests

- segmentation rules
- evidence span validation
- playbook retrieval

12.3 Integration Tests

- end-to-end pipeline run with fixed LLM stubs (for determinism)

12.4 Human QA

- sample outputs reviewed by privacy-literate reviewer

---

13. Deployment (MVP)

13.1 Suggested Stack

- Frontend: Vercel or containerized hosting
- Backend + worker: containerized (Kubernetes optional; can start with simpler deployment)
- Postgres managed
- Redis managed
- S3-compatible storage

13.2 Secrets

- Managed secrets store
- Rotate keys; least-privilege IAM

---

14. Key Design Decisions (MVP)

1) **Evidence validation is deterministic** (span-based). No exceptions. 2) **Playbook is the policy source**; LLM is a structured interpreter. 3) **Light RAG only** (playbook retrieval), not general internet or broad corpora. 4) Async pipeline via queue; API remains responsive.

---

15. Open Questions

1) Exact extraction libraries for PDF (tradeoffs of layout vs speed) 2) Retention defaults (e.g., 30/90/180 days) and deletion UX 3) Whether to store full extracted text in DB vs object storage (cost/security) 4) Whether to include optional LLM verifier in MVP or Phase 1.5 5) Authentication approach (magic link vs OAuth vs SSO later) 6) Minimal progress reporting (polling vs SSE)

---

16. Appendix — Clause Types (MVP Enum)

- ROLES
- SUBJECT_DURATION
- PURPOSE_NATURE
- DATA_CATEGORIES_SUBJECTS

- SECURITY_TOMS
- SUBPROCESSORS
- TRANSFERS
- BREACH_NOTIFICATION
- DSAR_ASSISTANCE
- DELETION_RETURN
- AUDIT_RIGHTS
- CONFIDENTIALITY
- LIABILITY
- GOVERNING_LAW
- ORDER_OF_PRECEDENCE