📘 Blueprints: AI Knowledge Management System & AI Personal Finance Assistant (2025-ready)

This document provides **production-grade blueprints** for two projects blending **core AI/ML** with **full-stack** development. Each section includes: - System architecture (with diagrams) - Data model / database schema - ML model choices & training/evaluation pipelines - API design (endpoints, request/response, auth) - Infra/DevOps, observability, security & compliance - Testing strategy & rollout plan

> Suggested baseline stacks: **Next.js + Tailwind**, **FastAPI** (Python) or **Node.js/Express**, **PostgreSQL**, **Redis**, **S3-compatible object storage**, **vector DB (Pinecone/Weaviate/Milvus)**, **Docker + Kubernetes**, **OpenTelemetry**, **GitHub Actions**.

---

1) AI-Powered Knowledge Management System (RAG + Collaboration)

1.1 Product scope

- Upload & ingest PDFs, DOCX, HTML, audio/video (ASR), and websites.
- Automatic **chunking → embedding → indexing** in a vector DB.
- **Semantic search** + **RAG chat** with source citations & page anchors.
- **Auto-tagging/topic extraction**, custom taxonomies, collection/workspace management.
- Access control: orgs, workspaces, roles (admin, member, guest).
- Analytics: search quality, doc coverage, latency, usage.

1.2 High-level architecture

```
flowchart LR
  subgraph Client
    A[Next.js Web App]
  end
  subgraph Edge
    CDN[CDN/Edge Cache]
  end
  subgraph Backend
    B[API Gateway]
    C[Auth Service (OIDC/JWT)]
    D[Ingestion Service]
    E[Processing Queue]
    F[Embeddings Service]
    G[RAG/Chat Service]
    H[Search Service]
    I[Admin/Analytics Service]
  end
  subgraph Data
```

```
    J[(PostgreSQL)]
    K[(Object Store S3)]
    L[(Vector DB)]
    M[(Redis Cache)]
    N[(Elasticsearch/Opensearch)]
  end
  A-->CDN-->B
  B-->C
  B-->D
  D-->K
  D-->E
  E-->F
  F-->L
  B-->H-->L
  H-->N
  B-->G
  G-->L
  G-->J
  B-->I
  I-->J
  B-->J
  B-->M
```

1.3 Data model (ERD)

```
erDiagram
  USER ||--o{ MEMBERSHIP : has
  ORG ||--o{ WORKSPACE : has
  WORKSPACE ||--o{ DOCUMENT : contains
  DOCUMENT ||--o{ CHUNK : split_into
  DOCUMENT ||--o{ FILEVERSION : versions
  CHUNK ||--|| EMBEDDING : represented_by
  SEARCHQUERY ||--o{ SEARCHRESULT : yields
  CONVERSATION ||--o{ MESSAGE : has
  MESSAGE ||--o{ CITATION : cites

  USER {
    uuid id PK
    text email
    text name
    text auth_provider
    jsonb profile
    timestamptz created_at
  }
  ORG {
    uuid id PK
    text name
    timestamptz created_at
  }
  WORKSPACE {
```

```
  uuid id PK
  uuid org_id FK
  text name
  jsonb settings
}
MEMBERSHIP {
  uuid id PK
  uuid user_id FK
  uuid org_id FK
  text role  // owner|admin|member|guest
}
DOCUMENT {
  uuid id PK
  uuid workspace_id FK
  text title
  text source_type // pdf,docx,url,audio,video
  text storage_key // S3 key
  jsonb metadata // author, language, etc.
  timestamptz created_at
  timestamptz indexed_at
}
FILEVERSION {
  uuid id PK
  uuid document_id FK
  int version
  text storage_key
  text checksum
  timestamptz uploaded_at
}
CHUNK {
  uuid id PK
  uuid document_id FK
  int chunk_index
  text content
  int token_count
  jsonb meta // page, section, headings
}
EMBEDDING {
  uuid id PK
  uuid chunk_id FK
  vector(1536) vec // DB-specific vector type
  text model_name
}
SEARCHQUERY {
  uuid id PK
  uuid user_id FK
  text query
  jsonb filters
  timestamptz created_at
}
SEARCHRESULT {
```

```
  uuid id PK
  uuid searchquery_id FK
  uuid chunk_id FK
  float score
}
CONVERSATION {
  uuid id PK
  uuid workspace_id FK
  uuid user_id FK
  jsonb system_prompt
  timestamptz created_at
}
MESSAGE {
  uuid id PK
  uuid conversation_id FK
  text role // user|assistant|system
  text content
  jsonb tool_calls
  timestamptz created_at
}
CITATION {
  uuid id PK
  uuid message_id FK
  uuid document_id FK
  int page
  float score
  text snippet
}
```

1.4 ML components

- **Text chunking:** recursive character splitter by headings/page boundaries (target 300–800 tokens).
- **Embeddings:** `text-embedding-3-large` or `bge-large` (1024–1536 dims). Store in vector DB with IVF/HNSW index; cosine similarity.
- **Reranking (optional):** Cross-encoder (e.g., `cross-encoder/ms-marco-MiniLM-L-6-v2`).
- **Summarization/RAG generation:** LLM (Llama 3.x, Mixtral, or GPT-4o-mini) with **structured citations** and **grounding checks**.
- **Auto-tagging/topic extraction:** Zero-shot or weakly-supervised classifier using embeddings + keyword extraction (YAKE/KeyBERT) + LLM refinement.
- **ASR for audio/video:** Whisper-medium/large; diarization with Pyannote.
- **Evaluation:**
- **Retrieval:** nDCG@k, Recall@k using labeled query–passage pairs.
- **RAG QA:** Faithfulness, answer relevancy (LLM-as-judge) + human spot-checks.
- **Latency SLOs:** P50 < 1.5s for search, P95 < 5s for chat.

1.5 Pipelines

- **Ingestion:** `Upload → Virus scan → OCR (Tesseract) → Chunk → Embed → Upsert` (idempotent; versioned).

- **Search:** `Query → Embed → ANN retrieve top-k → (optional rerank) → return chunks + highlights`.
- **Chat (RAG):** `User prompt → Search top-k → Build context (de-dup by doc) → LLM → citations → safety/redaction`.
- **Feedback loop:** Thumbs up/down stored to tune chunking and reranker.

1.6 API design (key endpoints)

Auth - `POST /auth/login` (OIDC code exchange) → JWT access + refresh.

Documents - `POST /workspaces/{id}/documents` (multipart) → `{document_id}` - `GET /documents/{id}` → document metadata - `GET /documents/{id}/chunks` → paginated chunks - `POST /documents/{id}/reindex` → re-run pipeline

Search & RAG - `POST /search` `{ query, workspace_id, filters?, k? }` → `[ {chunk_id, doc_id, score, snippet, page} ]` - `POST /chat` `{ conversation_id?, prompt, workspace_id }` → stream SSE/WS `{token, citations[]}`

Taxonomy/Tags - `POST /workspaces/{id}/tags` `{name, color?}` - `POST /documents/{id}/tags` `{tag_ids: []}`

Analytics - `GET /analytics/usage` (org/workspace scoped) - `GET /analytics/search` (top queries, zero-hit queries)

Errors: JSON problem details (RFC 7807).

1.7 Deployment & infra

- **Containers:** Separate deploy units per service; autoscale RAG/embedding workers.
- **Queues:** RabbitMQ/SQS for ingestion + async jobs.
- **Storage:** S3 with lifecycle + Glacier; signed URLs on download.
- **Observability:** OpenTelemetry → Tempo/Jaeger (traces), Prometheus (metrics), Loki (logs), Grafana dashboards.
- **Feature flags:** ConfigCat/Unleash.

1.8 Security & compliance

- Row-level security in Postgres (workspace/org scoping).
- KMS-managed encryption at rest; TLS in transit; secrets via Vault/SSM.
- PII redaction in RAG outputs (NER-based + rule-based).
- Access logs, audit trails (who viewed what doc/chunk).

1.9 Testing & QA

- **Unit:** chunkers, embedding adapters, LLM prompt functions.
- **Integration:** end-to-end ingestion (synthetic docs).
- **Load:** k6 for search/chat throughput.
- **Eval harness:** golden queries for retrieval & RAG.

1.10 Milestone roadmap (suggested)

1. M1: PDF ingestion → chunks → embeddings → semantic search.
2. M2: RAG chat with citations.
3. M3: Auto-tagging + analytics.
4. M4: Multi-tenant, RBAC, audit.
5. M5: ASR, web page crawler, reranking.

---

2) AI Personal Finance Assistant (Mobile + Backend + ML)

2.1 Product scope

- Connect bank accounts (Plaid/Salt Edge) and import transactions.
- Auto-categorize expenses; merchant normalization.
- Cashflow forecasting & bill prediction; budget tracking.
- Fraud/anomaly alerts; personalized savings goals & tips.
- Cross-platform mobile app (React Native/Flutter) + web dashboard (Next.js).

2.2 High-level architecture

```
flowchart LR
  A[Mobile App / Web]-->B[API Gateway]
  B-->C[Auth Service]
  B-->D[Bank Link Service]
  B-->E[Transaction Service]
  B-->F[ML Service]
  B-->G[Notification Service]
  B-->H[Budget/Goals Service]
  B-->I[Analytics Service]
  subgraph Data
    J[(PostgreSQL)]
    K[(Object Store)]
    L[(Feature Store)]
    M[(Redis Cache)]
    N[(Event Bus/Queue)]
  end
  D-->N
  E-->J
  F-->L
  F-->J
  G-->J
  I-->J
  B-->M
```

2.3 Data model (ERD)

```
erDiagram
  USER ||--o{ ACCOUNT : owns
```

```
ACCOUNT ||--o{ TRANSACTION : has
USER ||--o{ BUDGET : sets
USER ||--o{ GOAL : defines
TRANSACTION ||--o{ ALERT : triggers

USER {
  uuid id PK
  text email
  text name
  text country
  text kyc_status
  timestamptz created_at
}
ACCOUNT {
  uuid id PK
  uuid user_id FK
  text provider // plaid, saltedge, mock
  text external_id
  text name
  text type // checking, savings, credit, loan
  numeric balance
  text currency
  timestamptz linked_at
}
TRANSACTION {
  uuid id PK
  uuid account_id FK
  timestamptz posted_at
  text description
  text merchant_raw
  numeric amount // negative: debit, positive: credit
  text currency
  text category_pred
  float category_conf
  text category_final
  jsonb enrichment // location, mcc, geo
  text status // pending, posted
  text hash // idempotency
}
CATEGORY {
  uuid id PK
  text name
  text parent_id FK nullable
}
BUDGET {
  uuid id PK
  uuid user_id FK
  uuid category_id FK
  numeric monthly_limit
  timestamptz start_month
}
```

```
GOAL {
  uuid id PK
  uuid user_id FK
  text name
  numeric target_amount
  numeric current_amount
  timestamptz target_date
}
ALERT {
  uuid id PK
  uuid transaction_id FK
  text type // fraud, overspend, unusual
  text message
  bool read
  timestamptz created_at
}
```

2.4 ML components

- **Expense categorization:**
- Baseline: linear/LightGBM with features (n-grams of `description`, merchant embeddings, MCC, amount bins, day-of-week).
- Advanced: Transformer text encoder (hf: `MiniLM` / `distilbert`) + tabular features → shallow classifier / logistic head.
- Online learning with user corrections → label store; active learning sampling.
- **Merchant normalization:** nearest-neighbor over merchant embeddings + rules (regex, stopwords).
- **Cashflow & bill prediction:**
- Identify recurring patterns via periodogram + sequence alignment; forecast via Prophet or TFT/LSTM.
- Per-user models with hierarchical fallback (global → segment → user).
- **Fraud/anomaly detection:** Isolation Forest/LOF on transaction features; threshold by account type.
- **Recommendation engine:** bandit (LinUCB/Thompson) for tips/offers.
- **Evaluation:** macro-F1 for categorization; MAE/MAPE for cashflow; precision@k for alerts; A/B for tips.

2.5 Feature/label pipelines

- **Ingestion:** Webhooks from provider → `TransactionService` upserts; dedupe by `hash`.
- **Feature store:** Feast-like schema with `merchant_embedding`, `amount_norm`, `dow`, `rolling_30d_spend`, `budget_utilization`.
- **Training loop:** nightly batch (Spark/SQL) + periodic online updates from user feedback.
- **Model registry:** MLflow; versioned artifacts; canary deploy via shadow predictions.

2.6 API design (key endpoints)

Auth & users - `POST /auth/login` → tokens - `GET /me` → profile, KYC status

Bank linking - `POST /bank/link/token` → provider link token - `POST /bank/link/exchange {public_token}` → accounts - `GET /accounts` → list

Transactions - `GET /accounts/{id}/transactions?from&to&limit&cursor` - `POST /transactions/{id}/categorize` `{category_final}` (stores user override) - `GET /insights/spend-summary?period=month` → totals by category

Forecast & alerts - `GET /forecast/cashflow?days=30` - `GET /alerts` → fraud/overspend alerts

Budgets & goals - `POST /budgets` `{category_id, monthly_limit}` - `GET /budgets` → with utilization - `POST /goals` `{name, target_amount, target_date}` - `GET /goals` → progress

Notifications - Webhooks → push/SMS/email via provider; rate-limited, idempotent.

2.7 Deployment & infra

- **Providers:** Sandbox during dev; abstract provider interface.
- **Workers:** Categorization, forecasting, and alerting as async jobs (Celery/RQ/Kafka consumers).
- **Caching:** Redis for hot dashboards; TTL 60–300s.
- **Observability:** tracing for provider calls; SLOs: P95 transactions list < 800ms; forecast < 2s.

2.8 Security & compliance

- **Secrets:** vault/SSM; never store bank credentials; token exchange only.
- **PII:** field-level encryption (pgcrypto) for names/emails; row-level tenancy.
- **Compliance prep:** PCI-DSS boundary (avoid card PAN); SOC2 logging; data retention policies.

2.9 Testing & QA

- **Synthetic data generator** for transactions (seasonality + anomalies).
- **Golden sets** for categorization (stratified by merchant & amount).
- **Contract tests** against provider sandbox.
- **A/B framework** to evaluate tips & forecast UIs.

2.10 Milestone roadmap

1. M1: Provider link + transactions import + basic categorization.
2. M2: Budgets, goals, spend summaries.
3. M3: Cashflow & bill prediction + alerts.
4. M4: Merchant normalization + feedback learning.
5. M5: Web dashboard, analytics & multi-tenant admin.

---

3) Shared implementation notes (both projects)

3.1 Tech choices

- **Frontend:** Next.js (App Router), React Query, shadcn/ui, TanStack Table, Tailwind; SSE/WS for streaming.
- **Backend:** FastAPI with Pydantic v2; Uvicorn; or Node/Express + Zod; modular service boundaries.
- **DB:** PostgreSQL 15+ with pgvector if self-hosted embeddings; TimescaleDB for time series (finance).
- **Vector DB:** Pinecone/Weaviate/Milvus; HNSW index; replicas=2.

3.2 Patterns

- **Idempotency keys** on writes.
- **Outbox pattern** for reliable events.
- **Saga/compensation** for multi-step flows (bank link, ingestion).

3.3 Sample infra as code (Terraform modules)

- VPC, subnets, SGs, RDS Postgres, Elasticache Redis, EKS cluster, S3 buckets with lifecycle, IAM roles with least privilege.

3.4 Prompts & safety

- Guardrails for LLM: system prompts enforce citations (KM) and no financial advice guarantees (Finance).
- PII redactors before logging; structured logs only.

---

4) Example payloads

4.1 Knowledge Management – `/search`

```
{
  "query": "How does the SLA define uptime?",
  "workspace_id": "7b5c...",
  "filters": {"tags": ["SLA", "contracts"], "date_gte": "2025-01-01"},
  "k": 8
}
```

Response

```
[
  {"doc_id":"a12f...","chunk_id":"c9f...","page":3,"score":0.83,
   "snippet":"Service uptime is defined as ... excluding planned
maintenance ..."}
]
```

4.2 Finance – `/forecast/cashflow`

```
{
  "days": 30
}
```

Response

```
{
  "daily": [
```

```
    {"date":"2025-08-15","projected_balance": 1234.55},
    {"date":"2025-08-16","projected_balance": 1189.10}
  ],
  "expected_bills": [
    {"name":"Rent","date":"2025-09-01","amount":-800.00}
  ],
  "confidence": 0.74
}
```

---

5) What to build first (practical cut)

- **KM:** start with PDF ingestion → chunk/emb → semantic search; add RAG chat once search quality is stable.
- **Finance:** start with sandbox transactions → categorization with feedback UI → budgets; then forecasting + alerts.

Need a **starter repo structure** (backend + frontend + infra) or **Mermaid ERD exported as SQL**? Tell me your preferred stack (FastAPI vs Node, Pinecone vs pgvector), and I'll generate production-quality scaffolding next.