Technical Decisions — E-Commerce Pets Backend

1) Objective and Scope

Problem to solve: Inventory and sales of pet products; authentication and authorization; carts and checkout.

Scope of the service: REST API (Flask) with PostgreSQL, Redis caching, JWT, and tests.

Included modules: auth (login/register/JWT), users, products, carts, orders. Out of scope: real payment gateways, invoicing, logistics, UI panels.

2) Architecture and Stack

Framework: Flask (microframework → explicit control, simple learning curve).
Database: PostgreSQL (strong consistency, advanced types, native ENUM).
ORM: SQLAlchemy with Alembic migrations.
Cache: Redis (low latency, TTL per key, O(1) ops).
Auth: JWT (stateless, easy to scale horizontally).
Schema: ECommercePets (Postgres namespace).
Environment: venv, requirements.txt.

Blueprints: separated into auth, products, carts, orders for maintainability and testability.

3) Data Modeling (executive summary)

- Users: id, unique email, password_hash (bcrypt), role (ADMIN/CUSTOMER).
- Products: transactional stock; immutable vs mutable fields (price/stock).
- Cart/Items: status (OPEN/CONVERTED/ABANDONED), N–M relation product-cart with quantity.
- Orders/Transactions: derived from cart at checkout; idempotency enforced.

Key decisions: ENUMs for roles/status, indexes on email/name, composite PK for cart_item, FKs with ON DELETE CASCADE, created_at/updated_at for audit.

4) Authentication and Authorization (why JWT)

JWT vs server sessions: JWT is stateless, avoiding central session storage and supporting horizontal scaling.
Custom claims: sub (user id), role, exp short; no sensitive data.
Role-based auth: decorator @role_required('ADMIN', ...).
Rotation/expiration: short-lived access tokens; optional refresh tokens.
Logout/deny-list: jti stored in Redis with TTL until expiration; revoked tokens are blocked.

5) Read Caching and TTL (what, why, how)

Objective: reduce DB latency and load for heavy-read endpoints (GET /products, GET /products/{id}).
Key design: products:list:{query-hash}, products:id:{id}.
TTL: 120–300 seconds. Rationale: product catalog changes rarely compared to read frequency, so a 2–5 min TTL balances freshness vs performance.
Invalidation: on product changes or checkout, affected keys and lists are deleted.
Justification: TTL ensures auto-recovery even if invalidation fails.

9) Observability

Structured logging at INFO/ERROR, correlated by request id.
Optional metrics: latency per endpoint, cache hit ratio, error rates.

10) Testing

Unit tests for pure services/validators.
Integration tests for endpoints with test DB.
Fixtures: admin & customer users, products, JWT tokens.
Execution: pytest -q (covered in README).

12) Risks and Future Work

Backlog: rate limiting, cursor-based pagination, auditable soft-delete, post-checkout webhooks.
Hardening: password policies, optional MFA, JWT key rotation, deny-list in incidents.

Annex: DB Diagrams

See Diagram_ECommercePets.pdf (not included in this document).