Software Requirements Specification (SRS)

Product (working title): ShopPilot — AI Co-Founder for Indie Stores
Version: v0.1 (Sprint 2 submission)
Owner: E-commerce & FinTech Stream (Code Integration Lead: Daniel Le)

---

1. Introduction

Purpose. Define a buildable scope for an AI-assisted ecommerce companion that helps a solo/family retailer create segments, draft content, run campaigns, recover carts, and review performance. The system uses multiple AI agents coordinated by an **MCP server** and **n8n** workflows, with a simple dashboard and optional chat.

Goals for this submission. Deliver a formal specification plus a runnable **local prototype** (no paid infra) to demonstrate end-to-end value on mock/Shopify-dev data.

---

2. Scope

- Integrate with **Shopify** (dev store) or **CSV** imports as a fallback.
- Use **Klaviyo** for email/SMS delivery in the prototype.
- Orchestrate side-effects through **n8n** (Docker), keeping AI reasoning in a separate **MCP server**.
- Provide a minimal dashboard (tiles) and a chat panel for one-click/typed actions.
- Ship a **Cart-Recovery MVP** flow and foundations for campaigns/segments.

Out of scope (Sprint 2): multi-tenant billing, advanced RBAC UI, production SSO.

---

3. Users & Roles

- **Owner** – full access, approves spending and sends; sees analytics.
- **Marketer** – drafts templates, creates segments, schedules campaigns.
- **Analyst** – reads insights/attribution; cannot send.
- **Viewer** – read-only.

---

4. High-Level System Overview

```
Frontend (React/Next) ──> Backend API (Node/Express)
   │                              │
   │  Chat / Tiles                ├──> MCP Client ↔ MCP Server (FastAPI)
```

```
   │                              │                          │
   └──────────────> Auth, CRUD, Jobs│                          └──> Tools
(JSON-typed):
                                  │                               segment.buildRFM
                                  │                               cart.analyze
                                  │                               template.generate
                                  │                               email.compose /
sendBulk
                                  │                               job.getStatus /
report.campaign
                                  │
                                  └──> n8n (Docker) ──> Shopify / Klaviyo / CSV
```

LLMs: Ollama local models (`qwen2.5:7b`, `llama3.1:8b`) for analysis/chat; optionally OpenAI if key provided.

---

5. Functional Requirements

5.1 Authentication & Settings

- Email/password login (local dev); roles: Owner/Marketer/Analyst/Viewer.
- Connectors: Shopify API keys (or CSV upload), Klaviyo API key, store timezone, send caps, quiet hours.

5.2 Data Sync

- Pull customers, carts, orders, products from Shopify dev (or import CSV).
- n8n schedules sync; dedupe/upsert; capture consent flags.

5.3 Segmentation

- Build/save segments using filters (RFM, spend, last seen, SKU tags).
- Preview segment size; export to Klaviyo list.

5.4 Templates & Content

- Template library (HTML + variables like `{{first_name}}`, `{{discount_code}}`).
- AI assists: subject line, tone, CTA variants; image alt-text suggestions.

5.5 Campaigns & Automations

- Broadcast: choose segment + template + schedule; optional A/B on subject/time.
- Automations library: Abandoned Cart (MVP), Win-back, Post-purchase (stubs).
- Confirm step with dry-run preview (sample recipients; estimated sends).

5.6 Chat + MCP Tooling

- Chat can **list tools**, generate templates, propose segments, and create a **campaign draft**; requires explicit **CONFIRM** to send.
- Each tool has a schema, pre-conditions, and returns structured results saved to an audit log.

5.7 Analytics & Reporting

- Delivered/open/click/conversion/revenue by campaign and flow.
- Cart-recovery revenue attribution; per-segment lift.
- Send-time heatmap; subject leaderboard (basic).

5.8 Audit & Safety

- Log every tool call and workflow execution (actor, input hash, outcome).
- Consent/opt-in status, suppression lists, easy unsubscribe.
- Quiet hours; global daily cap; soft killswitch to stop sends.

---

6. Non-Functional Requirements

- **Performance:** P95 API < 400 ms (reads); enqueue bulk send < 60 s.
- **Availability:** 99.5% target for the dashboard (prototype: best-effort).
- **Security:** secrets in `.env`; HTTPS in production; role-based auth.
- **Privacy:** GDPR/CAN-SPAM style principles; delete/export on request.
- **Observability:** structured logs; n8n job metrics; simple health checks.

---

7. External Interfaces

- **Shopify API** (customers, carts, orders, products).
- **Klaviyo API** (lists/segments, email/SMS send, metrics).
- **n8n** (webhooks to trigger flows; job status via HTTP).
- **Ollama** (`/api/generate`, optional `/api/chat`).

---

8. Data Model (Simplified)

```
User(id, role, email, hashed_password)
Store(id, name, timezone, connectors)
Customer(id, email, phone, consent_email, consent_sms, rfm_scores, last_seen_at)
Product(id, title, price, tags)
Cart(id, customer_id, items[], updated_at)
Segment(id, name, filter_json, size_estimate)
Template(id, name, html, variables[])
```

```
Campaign(id, name, type, segment_ids[], template_id, status, schedule_at,
metrics)
Job(id, type, status, started_at, finished_at, error)
AuditLog(id, actor, action, tool, input_hash, output_summary, timestamp)
```

---

9. MCP Tools (Example Specs)

`segment.buildRFM({ lookbackDays, thresholds }) -> { segmentId, size }`
`cart.analyze({ lookbackDays? }) -> { topReasons[], stats }`
`template.generate({ goal, tone, length, variables }) -> { subject, html }`
`email.compose({ segmentId, templateId }) -> { campaignDraftId }`
`email.sendBulk({ campaignDraftId, scheduleAt?, dryRun? }) -> { jobId }`
`job.getStatus({ jobId }) -> { status, counts, errors[] }`
`report.campaign({ campaignId }) -> { metrics, attribution }`

Each tool validates input, checks permissions, and triggers **n8n** for side-effects; returns a `jobId` for polling.

---

10. Constraints & Assumptions

- Ollama models run locally and are reachable by the backend.
- n8n runs in Docker at `http://localhost:5678`; exposed via ngrok if needed.
- For non-Shopify stores, CSV import or custom webhook events are accepted.

---

11. Acceptance Criteria (MVP)

- Connectors configured; import ≥ 200 mock customers.
- Create one segment; generate a template via AI; perform a **dry-run** preview.
- Send a real test campaign to a small list via Klaviyo and display metrics.
- Chat can **list tools**, produce a campaign draft, and ask for **CONFIRM** before sending.

---

12. Risks & Mitigations

- **Integration complexity / slow velocity** → pair programming; smaller task slices; visible board.
- **API limits / failures** → queues, retries, DLQ; synthetic tests.
- **Privacy concerns** → consent flags, suppression, HITL, data minimisation.
- **Team bandwidth** → daily 15-min stand-ups; early escalation to PO.

---

13. Roadmap (Next 2–3 Sprints)

- **Sprint 2**: Local MVP (cart recovery); Segments + Templates basics; MCP tools: `template.generate`, `email.sendBulk`, `job.getStatus`.
- **Sprint 3**: RFM segmentation UI; A/B subject lines; analytics read-back; consent management; audit viewer.
- **Sprint 4**: Inventory/Order agent; SEM basics; dashboard polish; deploy to a small VPS.

Prototype Implementation Guide (Local, No Paid Infra)

Goal: A runnable demo: mock/Shopify-dev abandoned cart → AI-drafted message → Klaviyo test send → metrics visible.

A) Prerequisites

- **Docker Desktop**, **Node 18+**, **Python 3.10+**, **ngrok**.
- **Ollama** with models: `ollama pull qwen2.5:7b` and `ollama pull llama3.1:8b`.
- **Klaviyo** free account (API key).
- **Shopify** dev store (optional for real webhooks).

B) Repository Layout

```
ai-org-design/
  prototype/
    mcp-server/        # FastAPI + tool schemas
    cart-recovery/     # n8n workflows + mock data + scripts
  diagrams/
  ui-wireframe/
  docs/
```

C) Start n8n (Docker)

```
docker run -it --name n8n -p 5678:5678
  -e N8N_BASIC_AUTH_ACTIVE=true
  -e N8N_BASIC_AUTH_USER=admin
  -e N8N_BASIC_AUTH_PASSWORD=pass
  n8nio/n8n
```

Open `http://localhost:5678` and create a workflow:
Webhook (`/webhook/shopify/cart`) → **HTTP Request** (call MCP tool) → **Klaviyo** (send email) → **Respond to Webhook** (200).

D) Minimal MCP Server (FastAPI)

`prototype/mcp-server/app.py`

```
from fastapi import FastAPI
from pydantic import BaseModel
import requests, os

app = FastAPI()

class TemplateGen(BaseModel):
    goal: str
    tone: str = "friendly"
    length: str = "short"
    variables: dict

@app.get("/tools")
def list_tools():
    return {"tools": ["template.generate", "email.sendBulk", "job.getStatus"]}

@app.post("/tool/template.generate")
def template_generate(req: TemplateGen):
    # Call local Ollama to generate subject/body (simplified)
    prompt = f"Goal: {req.goal}. Tone: {req.tone}. Include variables:
{list(req.variables.keys())}."
    # Pseudo: call ollama here; replace with real HTTP call in your env
    subject = "We saved your cart ✨"
    html = f"<h1>Don't miss out, {req.variables.get('first_name','there')}!</
h1>"
    return {"subject": subject, "html": html}

class SendBulk(BaseModel):
    campaignDraftId: str
    scheduleAt: str | None = None
    dryRun: bool = False

@app.post("/tool/email.sendBulk")
def email_send_bulk(req: SendBulk):
    # Trigger n8n workflow by HTTP (map campaignDraftId to Klaviyo send)
    return {"jobId": "job_123"}

@app.get("/tool/job.getStatus")
```

```
def job_status(jobId: str):
    return {"status": "completed", "counts": {"sent": 120, "failed": 3}}
```

Run: `uvicorn app:app --port 8000 --reload`

E) n8n → MCP → Klaviyo wiring

- **HTTP Request** node: `POST http://localhost:8000/tool/template.generate` with JSON payload `{goal, tone, variables}`;
- Feed result into **Klaviyo** node (use subject/html from previous step).
- Save the run output to a file or a simple DB (optional) for metrics.

F) Optional: Simple Chat Frontend

- A tiny React page that calls `/tools` to list capabilities, then calls `/tool/*` endpoints via the backend.
- Show preview of 20 sample recipients before **CONFIRM**.

G) Evidence to Capture

- Screenshot of n8n workflow.
- Terminal logs from MCP server (tools called).
- Klaviyo test list showing one send.
- 30–45s screen recording of the end-to-end flow.

H) Acceptance Test (Demo Script)

1. Trigger a mock abandoned cart event (post JSON to n8n webhook).
2. n8n calls MCP `template.generate` → returns subject/html.
3. n8n calls Klaviyo **Send Email** to a test list.
4. Show sent email + MCP logs + n8n success.
5. Conclude with metrics panel (basic counts).

Appendix A — Sample YAML Workflow

```
workflow: cart_recovery_v1
trigger: shopify.checkout_abandoned
steps:
  - name: generate_template
    tool: template.generate
    params: { goal: "recover_cart", tone: "friendly", variables: { first_name:
"{{customer.first_name}}" } }
  - name: send_bulk
    tool: email.sendBulk
```

```
    params: { campaignDraftId: "draft_abc", dryRun: false }
  - name: poll_status
    tool: job.getStatus
    params: { jobId: "{{steps.send_bulk.result.jobId}}" }
```

Appendix B — Minimal API Sketch (Backend)

```
POST /api/segments {filters} → {segmentId, size}
POST /api/templates {html, subject}
POST /api/campaigns/draft {segmentId, templateId} → {campaignDraftId}
POST /api/campaigns/send {campaignDraftId, scheduleAt?, dryRun?} → {jobId}
GET  /api/jobs/:id → {status, counts}
GET  /api/analytics/campaign/:id → {metrics}
```

Submission note. This pack includes a formal SRS and a practical prototype guide that aligns with PO direction, uses only local resources (Ollama + n8n Docker), and is intentionally small so it can be demoed quickly and iterated in Sprint 3.