

EXECUTIVE SUMMARY

This document defines the five modes of operation for The Quiet Feed, covering deployment environments (cloud vs. local), access tiers (ANONYMOUS, ENHANCE, HARD COPY), and interface types (web browser vs. CLI). Each mode has distinct characteristics for authentication, storage, LLM processing, and feature availability.

***Core principle:** The same codebase serves all modes. Environment variables and configuration determine behavior, not code branches.*

MODE OVERVIEW

MODE	ENVIRONMENT	TIERS	LLM	STORAGE
AWS Web	Production cloud	ANON / ENHANCE / HARD COPY	Claude API	DynamoDB + S3
Local Web	Developer workstation	ANON / ENHANCE	Ollama	Dynalite (ephemeral)
CLI	Terminal	ANON / ENHANCE	Ollama	Dynalite (ephemeral)

MODE 1: AWS WEB (PRODUCTION)

The primary deployment mode. Runs on AWS infrastructure with full scaling, persistence, and commercial LLM access. This is what users at thequietfeed.com experience.

Architecture

- **Frontend:** Static HTML/CSS/JS served via CloudFront from S3
- **API:** HTTP API Gateway v2 → Lambda (Node.js 20, ARM64)
- **Authentication:** AWS Cognito with Google OAuth (LinkedIn planned)
- **Storage:** DynamoDB (on-demand) for user data, S3 for content cache
- **LLM:** Anthropic Claude API (Haiku for scoring, Sonnet for analysis)
- **CDN:** CloudFront with WAF rate limiting

Access Tiers

TIER	ACCESS	FEATURES	LIMITS
ANONYMOUS	No login. Session/local storage only.	Cyberated public feed. SCORE visible. Basic filters. Write Mode.	Non-persistent. No history.
ENHANCE	Free, authenticated. Social login.	Personal feed. All filters. DEDUP. TRACK and persist preferences.	50 articles per day.
HARD COPY	Paid subscription (\$20/mo during launch).	Unlimited platforms. Export. API access.	Priority processing.

ANONYMOUS Storage

Anonymous users have no server-side identity. Their preferences (mutes, display settings) are stored in browser localStorage. This persists across sessions on the same device but does not sync across devices. If localStorage is cleared, preferences are lost.

```
localStorage keys:  
  
quietfeed_mutes: JSON array of muted terms/accounts  
  
quietfeed_prefs: JSON object with display preferences  
  
quietfeed_last_feed: String, last viewed feed identifier
```

ENHANCE Persistence

ENHANCE users authenticate via Cognito OAuth. Their user ID (Cognito sub) is HMAC-SHA256 hashed before storage. All preferences, mutes, feed configurations, and OAuth tokens are stored in DynamoDB with the hashed sub as partition key.

Scoring Limits (ENHANCE)

ENHANCE users are limited to 50 newly-scored articles per day. This is a soft cap enforced by a daily counter in DynamoDB. Items already in the score cache do not count against this limit. The limit resets at midnight UTC.

***Rationale:** At \$0.00025/item for Claude Haiku scoring, 50 items/day costs ~\$0.0125/day or ~\$0.38/month per ENHANCE user. This is sustainable at the free tier while encouraging upgrade to HARD COPY for heavy users.*

MODE 2: LOCAL WEB (DEVELOPMENT)

For developers running The Quiet Feed on their workstation. Uses local services that mirror production but require no cloud resources. All costs (compute, LLM) are borne by the developer's machine.

Architecture

- **Frontend:** Express.js serving static files on localhost:3000
- **API:** Same Lambda handlers via httpServerToLambdaAdaptor.js
- **Authentication:** Mock OAuth2 server (Docker) or skip auth
- **Storage:** Dynalite (in-memory DynamoDB-compatible)
- **LLM:** Ollama with local models (phi3:mini, mistral:7b)
- **Tunnel:** ngrok for OAuth callbacks (optional)

Starting Local Environment

```
npm start # All services
npm run data # Dynalite only
npm run auth # Mock OAuth2 only
npm run server # Express server only
ollama serve # LLM server (separate terminal)
```

Access Tiers (Local)

TIER	ACCESS	NOTES
ANONYMOUS	No login required	Same as AWS, uses localStorage
ENHANCE	Mock OAuth or real Cognito via ngrok	No daily limit (your compute)
HARD COPY	Not available	No payment processing locally

Ephemeral Storage

Dynalite runs in-memory. Data persists only while the process runs. When you stop `npm run data`, all DynamoDB data is lost. This is intentional for development: each test run starts fresh. For persistent local storage, we may add SQLite or file-based storage in a future desktop release.

Local LLM (Ollama)

Scoring uses Ollama with lightweight models. Quality is lower than Claude but sufficient for development and testing. No API key required, no cost per request.

MODEL	SIZE	RAM	QUALITY	USE CASE
phi3:mini	3.8B	4GB	★★★★■	CI/CD, fast iteration
mistral:7b	7B	6GB	★★★★■	Development, realistic scores
llama3.2:8b	8B	8GB	★★★★■	Production-like testing

Environment Variables

```
# .env.proxy (local development)

ENVIRONMENT_NAME=proxy

LLM_PROVIDER=ollama

LLM_MODEL=phi3:mini

LLM_BASE_URL=http://localhost:11434/v1
```

MODE 3: CLI (TERMINAL)

A command-line interface for The Quiet Feed, inspired by Claude Code. Text-first, keyboard-driven, scriptable. Same backend as Local Web but with a terminal UI instead of a browser.

Design Philosophy

***MU/TH/UR 6000 (Alien, 1979):** The Nostromo's shipboard AI. CRT monitors, monospace text, institutional tone. The terminal as a place of truth-seeking.*

***Claude Code (Anthropic, 2024–):** "It's like a Unix utility... the same way you would compose grep or cat." The CLI as first-class citizen, not afterthought.*

Installation

```
# Install globally via npm
npm install -g @quietfeed/term

# Or run from repository
npx @quietfeed/term
```

Basic Usage

```
qf # Interactive mode (vim-style nav)
qf feed # Show default feed
qf feed --source hackernews # Filter by source
qf feed --mute "crypto" # Exclude topic
qf feed --wire-mode # Normalized headlines
qf feed --json # JSON output for piping
qf auth # Opens browser for OAuth
```

Interactive Mode

Running `qf` without arguments enters interactive mode with vim-style keyboard navigation:

- **j/k**: Navigate up/down through items
- **Enter**: Open selected item in browser
- **s**: Toggle score details
- **m**: Mute current item's source
- **/**: Search/filter
- **r**: Refresh feed
- **q**: Quit

Piping and Composition

```
# High-score items only
qf feed --json | jq '.items[] | select(.score > 80)'

# Count items per source
```

```
qf feed --json | jq '.items | group_by(.source) | map({source: .[0].source, count:
length})'

# Export to CSV
qf feed --json | jq -r '.items[] | [.score, .title, .url] | @csv' > feed.csv
```

Implementation Options

The CLI can be implemented using existing terminal UI libraries:

LIBRARY	LANGUAGE	FEATURES	EXAMPLE
Ink	Node.js	React for CLI, components	Claude Code uses this
blessed	Node.js	ncurses-like, low-level	More control, more work
Textual	Python	Modern TUI framework	Rich widgets, CSS-like styling
Bubble Tea	Go	Elm-inspired TUI	Very fast, good for distribution

Recommendation: Start with Ink (Node.js) for consistency with the rest of the codebase. Consider Bubble Tea (Go) if we want a single-binary distribution later.

FEATURE AVAILABILITY MATRIX

This matrix shows which features are available in each mode and tier combination.

FEATURE	AWS ANON	AWS ENHANCE	AWS HARD	LOCAL	CLI
View curated feed	✓	✓	✓	✓	✓
SCORE display	✓	✓	✓	✓	✓
SHIELD (no autoplay)	✓	✓	✓	✓	N/A
MUTE (basic)	✓ (local)	✓	✓	✓	✓
WIRE MODE	✓	✓	✓	✓	✓
Personal feed	—	✓	✓	✓	✓
DEDUP	—	✓	✓	✓	✓
TRACE	—	✓	✓	✓	✓
Persisted prefs	—	✓	✓	—	—
Multi-platform OAuth	—	✓	✓	Mock	Browser handoff
Export	—	—	✓	✓	✓
API access	—	—	✓	✓	✓
Daily score limit	—	50/day	Unlimited	Unlimited	Unlimited
LLM quality	Claude	Claude	Claude	Ollama	Ollama

AUTHENTICATION FLOWS

AWS Web (Production)

1. User clicks "ENHANCE" → redirects to Cognito hosted UI
2. User authenticates with Google (or LinkedIn, planned)
3. Cognito redirects to /auth/callback with auth code
4. Frontend exchanges code for JWT via /api/v1/auth/token
5. JWT stored in localStorage, sent with subsequent requests

Local Web (Development)

1. Mock OAuth2 server runs in Docker on localhost:8080
2. User clicks "ENHANCE" → redirects to mock OAuth
3. Mock server auto-approves, redirects with auth code
4. Same token exchange flow as production

5. For real OAuth, use ngrok tunnel for callbacks

CLI (Terminal)

1. User runs `qf auth`
2. CLI opens system browser to OAuth URL
3. User completes OAuth in browser
4. Callback redirects to local HTTP server (started by CLI)
5. CLI receives token, stores in `~/.quietfeed/credentials`
6. Subsequent `qf` commands use stored token

DATA FLOW BY MODE

AWS Web: Score Request

Browser → CloudFront → API Gateway → Lambda → DynamoDB (cache check)
↓ (cache miss)
Lambda → Claude API → DynamoDB (cache write) → Response

Local Web: Score Request

Browser → Express → Lambda handler → Dynalite (cache check)
↓ (cache miss)
Handler → Ollama → Dynalite (cache write) → Response

CLI: Feed Request

qf feed → HTTP to localhost:3000 → Same as Local Web
OR
qf feed → Direct function call (embedded server mode)

CONFIGURATION

Environment Files

FILE	MODE	PURPOSE
.env.prod	AWS Web (production)	Real Cognito, Claude API, prod DynamoDB
.env.ci	AWS Web (CI)	Real AWS, ephemeral stacks
.env.proxy	Local Web	Mock OAuth, Ollama, Dynalite
.env.proxyRunning	Local Web	Connect to already-running services
.env.test	Unit tests	Mocked services, no network

Key Variables

```
# LLM Configuration
LLM_PROVIDER=anthropic|ollama|openai
LLM_MODEL=claude-3-haiku-20240307|phi3:mini|gpt-4o-mini
LLM_BASE_URL=http://localhost:11434/v1
ANTHROPIC_API_KEY=sk-ant-...

# Storage
BUNDLE_DYNAMODB_TABLE_NAME=quietfeed-prod-bundles
SCORES_DYNAMODB_TABLE_NAME=quietfeed-prod-scores

# Feature Flags
ENHANCE_DAILY_LIMIT=50
```

ENABLE_LINKEDIN_OAUTH=false

FUTURE: DESKTOP APPLICATION

A potential future mode: a native desktop application with persistent local storage, offline capability, and optional cloud sync. Not in current scope but architecture should not preclude it.

Potential Architecture

- **Framework:** Electron or Tauri (Rust + WebView)
- **Storage:** SQLite for persistence, sync to cloud optional
- **LLM:** Ollama bundled or user-provided
- **Updates:** Auto-update via Electron/Tauri mechanisms

Design Considerations

The current Local Web mode is intentionally ephemeral (Dynamlite resets on restart). A desktop app would need persistent storage. The path forward:

1. Abstract data repositories (already done: dynamoDbXxxRepository.js)
2. Create sqliteXxxRepository.js implementations
3. Select repository at startup based on environment
4. Add sync layer for cloud backup (HARD COPY feature)

SUMMARY

MODE	WHO	WHEN	WHY
AWS Web	End users	Production	Full experience, real data
Local Web	Developers	Development	Fast iteration, no cloud cost
CLI	Power users, devs	Automation, scripting	Composable, keyboard-driven

The Quiet Feed. Multiple doors, same destination.