Developer Requirements Instagram ⇄ YouTube Reels/Shorts Forwarding Frontend + Backend Upgrade

Prepared for: Developer

Prepared by: Hardik Agrawal (product owner)

Purpose: Turn an existing Python backend automation (Reels ↔ Shorts) into a production-ready system with a clean single-user frontend, robust orchestration, monitoring, and easy per-connection control. The system must be deployable on a VPS using Docker and be architected so it can be converted to SaaS later.

---

1. Project Summary (one-liner)

Create a secure, Dockerized web application (frontend + backend) that connects existing Python automation scripts to a user-friendly admin UI to manage multiple connections that forward Instagram Reels → YouTube Shorts and YouTube Shorts → Instagram Reels; provide proxy management, token uploads, start/stop controls, retries, Telegram alerts, logging, and analytics.

---

2. High-level constraints & decisions (confirmed)

- Single user (basic login) now; design for future multi-tenant/SaaS extension.
- UI language: English.
- Backend language: Python (current scripts). Developer may refactor/rewrite as needed but must preserve existing behaviors/features.
- Hosting: VPS. Delivery may include Docker (recommended: multi-container Docker Compose).
- Storage: videos stored on server filesystem (no frontend upload storage). Optionally support S3-compatible later.
- Notification channel: Telegram channel (bot posts with mention details).
- Scale target: ~50–100 connections, concurrent forwarding throughput typical 10-15 concurrent jobs.
- Deliverable support: 3–4 day posting test window after delivery.

---

3. Architecture & technology recommendations (developer may choose alternatives but justify)

Recommended stack (preferred, but flexible if you justify alternatives):

- Backend API: **FastAPI** (Python) — async, performance, auto docs.
- Worker / Task queue: **Celery** with **Redis** (or RQ with Redis) for job queueing, retries, scheduling.
- Database: **PostgreSQL** (store metadata, tokens, accounts, connections, logs).

- Frontend: **React** (create-react-app / Vite) with component lib (Material UI or Tailwind). Must be responsive.
- Reverse Proxy: **Nginx** for HTTPS termination / static assets.
- Containerization: **Docker** with docker-compose.yml (separate containers: backend, frontend, db, redis, worker, nginx).
- Storage: Local disk for videos (configurable path); expose retention policy and cleaning job.
- Secrets: environment variables + encrypted DB fields. Optionally integrate HashiCorp Vault if desired.
- Authentication: Basic single-user login (username/password) with bcrypt and session/cookie-based auth or JWT.
- Real-time updates: WebSocket (Socket.IO / FastAPI WebSocket) or polling for job status.

Developer must deliver a Docker Compose deployment and a plain VPS deployment guide.

---

4. Functional requirements (core features)

4.1 User & Access

- Single user login (username/password). Password stored hashed (bcrypt). Admin settings page to update password.
- Session timeout configurable.

4.2 Account management (Accounts = Instagram / YouTube)

- CRUD for accounts:
    - Instagram account: name/alias, account identifier (username), method type (Graph API or current API), credentials/token references (not raw password in UI unless required by your current script), linked Facebook Page id (if Graph API).
    - YouTube account: alias, uploaded OAuth client JSON file (client_id, client_secret), refresh/access token metadata, channel id.
- Upload YouTube token JSON via frontend (store securely on server; show expiry timestamps notification in telegram channel).
- Show token expiry dates and send Telegram alerts when tokens are expired or near expiry give option for alert like send alert before : 2days.

4.3 Connections (Mapping rules between source and destination)

- Create connections with fields:
    - Connection name/alias.
    - Type: Insta -> YouTube or YouTube -> Insta.
    - Source account (select).

  - Destination account (select).
  - Proxy (optional; select from proxy pool or none).
  - Token references (if special per-connection).
  - Filter rules: forward all / only if caption contains X / only new posts / include old posts (existing script supports forwarding old reels — keep this option).
  - Schedule options: continuous (watch-and-forward), or scheduled windows for old reels.
  - Start/Stop/Pause controls per connection (single click).
  - Run now (manual trigger).
- Per-connection view should show recent jobs, success/fail counts, and last run timestamp.

4.4 Proxy manager

- Add/edit/delete proxies (support HTTP/HTTPS and SOCKS5).
- Option to mark proxy as per-account or global.
- Health check endpoint: test connection through proxy and show status last checked timestamp.
- Proxy usage: assign proxy to account or to connection; allow rotation policy configuration (round-robin, random).

4.5 Job & worker management

- Jobs enqueued per new source post or manual trigger.
- Automatic retry policy with exponential backoff (configurable number of retries — default 3).
- Failure handling: on failure send Telegram alert (with account, connection name, job id, error summary and solution to it). If retry succeeds, send success alert mentioning resolved.
- If CAPTCHA encountered: Pause job and send Telegram alert to channel with detailed message requesting manual resolution. Provide UI to mark captcha solved (manual intervention). Optionally support integration or investigation for third-party captcha solving services (2captcha / Anti-Captcha / CapMonster) include in dev notes.
- Provide dead-letter queue for permanently failed jobs with reason.

4.6 Video transforms & special features

- Must keep all existing features: upscaling and green-screen/greensreen layout option (available only to admin). These options must be toggleable per-connection.
- Automatic transcoding/compliance (auto-resize/format) to meet destination limitations if needed.

4.7 Token & credential expiry handling

- Detect and record expiry timestamps of tokens (YouTube refresh, Facebook access token).

- Send Telegram alert on expiry or on refresh failure, include app name, account names, connection name, and suggested action.
- If an access token is refreshed or re-uploaded, send success alert.

4.8 Logs, monitoring & analytics

- Centralized logs per job, per account, with searchable view (account, connection, error type).
- Analytics dashboard:
    - Monthly / weekly summary: total forwarded, successes, failures, success rate, per-account success rates.
    - Uptime / availability per account (percent of successful run).
    - Account failure reasons breakdown (token expired, captcha, proxy error, API error, network error rank most failure type to increase maximum uptime).
    - Retry statistics, average retries to success, dead-letter counts.
    - Top failing accounts, top failing connections.
- Export analytics to CSV.

4.9 Notifications

- Telegram channel alerts: all critical events posted to channel with structured message and tags (see examples below). Use your Telegram bot token & channel chat_id provided.
- On event resolution, post a follow-up message indicating resolved.
- Optionally support webhook endpoints for additional integrations.

---

5. Non-functional requirements

- **Security**: HTTPS enforced; secrets encrypted; password hashing (bcrypt); protect endpoints with authentication; restrict SSH access to VPS; logs scrub sensitive content.
- **Reliability**: use Celery + Redis for guaranteed task execution; idempotency for job processing; retries with exponential backoff; dead-letter handling.
- **Scalability**: design so new worker containers can be added; DB designed for scale (Postgres).
- **Maintainability**: code documented; REST API documented (OpenAPI / Swagger).
- **Observability**: health endpoints for backend, workers, Redis, DB; basic metrics (job queue length, jobs per minute).
- **Performance**: handle 4–10 concurrent jobs; design for horizontal scaling if needed.
- **Data retention**: logs retention policy and video retention configurable via admin.

---

6. Data model (core entities)

- **User**
    - id, username, password_hash,
- **Account**
    - id, type (instagram/youtube), alias, username, meta (json), token_ref, token_expiry, created_at, status
- **Connection**
    - id, name, type, source_account_id, dest_account_id, proxy_id, filters (json), transform_options (upscale/greenscreen), schedule, status (active/paused), created_at
- **Proxy**
    - id, host, port, type, username, password (encrypted), last_health_check, status
- **Job**
    - id, connection_id, source_post_id, status (queued/running/success/failed/captcha/paused), attempts, last_error, created_at, completed_at
- **Log**
    - id, job_id, level, message, timestamp, metadata
- **Analytics summary tables** (materialized daily/weekly):
    - forwarded_count, success_count, fail_count, avg_retries, top_errors

---

7. API endpoints (REST, examples)

Authentication:

- POST /api/login — { username, password } → { token/session }
- POST /api/logout

Accounts:

- GET /api/accounts — list
- POST /api/accounts — create (upload token JSON for YouTube as multipart)
- GET /api/accounts/{id}
- PUT /api/accounts/{id}
- DELETE /api/accounts/{id}

Connections:

- GET /api/connections

- POST /api/connections
- GET /api/connections/{id}
- PUT /api/connections/{id}
- DELETE /api/connections/{id}
- POST /api/connections/{id}/start
- POST /api/connections/{id}/stop
- POST /api/connections/{id}/run — manual run
- POST /api/connections/{id}/dry-run

Proxies:

- GET /api/proxies
- POST /api/proxies
- PUT /api/proxies/{id}
- DELETE /api/proxies/{id}
- POST /api/proxies/{id}/health-check

Jobs & Logs:

- GET /api/jobs?connection=&status=&from=&to=
- GET /api/jobs/{id}
- POST /api/jobs/{id}/retry
- GET /api/logs?account=&connection=&level=&from=&to=

Tokens:

- POST /api/tokens/upload — upload YouTube client JSON; returns token_id
- GET /api/tokens/{id}/status — expiry etc.

Analytics:

- GET /api/analytics/summary?period=monthly
- GET /api/analytics/account/{id}/metrics?period=weekly

Health:

- GET /api/health — returns health of services (DB/Redis/Worker)

Webhook:

- POST /api/webhook/telegram — for any external events (optional)

All endpoints must return structured JSON with standard error codes.

---

8. Telegram alert message formats (must be human-readable & contain details)

Format — Error / Alert

[APP-NAME] ⚠️ *ALERT*: <ALERT-TYPE>

Connection: <connection_name>

Source: <source_account_alias> (instagram)

Destination: <dest_account_alias> (youtube)

Time: <timestamp>

Job ID: <job-id>

Error: <short error summary>

Details: <short details or link to logs>

Action: <suggested action>

Example — Token expired

[SnapcoreForwarder] ⚠️ ALERT: ACCESS TOKEN EXPIRED

Connection: Insta->YouTube - "InstaNewsToYT"

Source: insta_news_account

Destination: my_youtube_channel

Time: 2025-10-16 20:48 IST

Token type: Facebook Graph API access token

Status: EXPIRED

Action: Re-authenticate in UI → Accounts → InstaNewsToYT

Example — Captcha

[SnapcoreForwarder] 🛑 CAPTCHA DETECTED (manual action required)

Account: insta_user_123

Connection: Insta->YouTube - "InstaNewsToYT"

Job ID: 8192

Time: 2025-10-16 20:50 IST

Desc: CAPTCHA encountered during login/upload. /solve not available.

Action: Open UI → Accounts → insta_user_123 → Click "Mark Captcha Solved" after manual solve.

Resolved

[SnapcoreForwarder] ✅ RESOLVED: CAPTCHA SOLVED

Account: insta_user_123

Connection: Insta->YouTube - "InstaNewsToYT"

Job ID: 8192

Time: 2025-10-16 20:55 IST

Bot must @mention the channel or include the channel chat_id (as required) and include a link to the job/log view.

---

9. Captcha handling & third-party solvers

- On CAPTCHA detection: immediately pause the account/connection, enqueue human-resolution alert to Telegram (detailed), and mark job state as captcha.
- Provide an admin UI button to mark "CAPTCHA solved" and resume jobs.
- Developer must research & propose third-party captcha solver integration (examples to evaluate: 2Captcha, Anti-Captcha, CapMonster). If integrated, make it a configurable option with cost/latency warnings. (Developer to present a short feasibility note.)

---

10. Retry & failure policy

- Retry attempts: default 3 retries, exponential backoff (e.g., 10s → 60s → 300s).
- On first failure: send Telegram alert (short).
- On each retry failure: send additional alert with attempt number.
- On final failure: mark failed and move to dead-letter queue. Provide UI to manually retry job.

---

11. Analytics & data analysis (detailed)

Provide the following analyses and visualizations in the Analytics dashboard:

Per-account & per-connection metrics

- Total forwarded (period)
- Success rate = successes / attempts
- Failure rate and top 5 failure reasons (token, captcha, proxy, API limit, network)
- Average retry count to success
- Uptime % (days with successful jobs / total days expected)

Operational dashboards

- Worker consumption / average worker utilization
- Proxy health overview (up / down, avg latency)
- Token expiry calendar (list tokens expiring in next 7/30 days)

Root-cause & drill-down

- Scatter of retries vs success (to detect brittle accounts)
- Correlate failure spikes to events (e.g., token renewals, proxy changes, code deploys)

Alerts / automated detection

- Threshold alerts (send Telegram): success rate < X% for an account over last 24h, token expiry within 72h, proxy error rate > Y% in last 10 jobs.
- Anomaly detection: sudden drop in success/uptime for account → auto-report top related logs.

Suggested visualizations

- Bar charts (top errors)

---

12. UI / UX flows (minimum screens)

1. **Login**
2. **Dashboard** — high-level metrics, active connections, quick actions (start/stop).
3. **Connections list** — create/edit/start/stop/run/dry-run.
4. **Connection wizard** — guided setup with required fields.
5. **Accounts** — add/edit accounts, upload YouTube JSON tokens, show token expiry and re-auth link.
6. **Proxies** — add/test/remove proxies; health checks.
7. **Jobs & Logs** — searchable job list, job details, logs, retry button.
8. **Analytics** — charts and tables described above.
9. **Notifications** — Telegram config, alert history.
10. **Settings** — retention policy, retry policy, worker settings, server paths.
11. **Admin utilities** — mark captcha solved, force-token-refresh, manual job enqueue.

---

13. Deployment & infrastructure checklist

- Provide docker-compose.yml with services: backend, frontend, db (postgres), redis, worker, nginx.
- Provide a production-ready Dockerfile for backend and frontend.
- Environment variable templates .env.example.
- Systemd service instructions or a simple ansible / bash script for one-click deploy + update.
- Healthcheck endpoints and Prometheus-compatible basic metrics (optional).

- Backup instructions for DB and token files.
- Disk space monitoring instructions & log rotation policy.
- Provide instructions for adding worker instances (scale out).

---

14. Deliverables (must include)

1. Full source code repository (Git) with commit history.
2. Dockerfiles + docker-compose.yml.
3. Deployment & server setup guide for VPS (Ubuntu recommended).
4. README & architecture diagram.
5. API docs (OpenAPI/Swagger).
6. Admin user manual (how to add accounts, connections, proxies, resolve captchas).
7. Test plan & test results (see acceptance tests).
8. 3–4 days post-deployment support for bug fixes related to deployment and core flows.
9. Optional: demo video (5–10 min) showing major flows.

---

15. Acceptance criteria & test cases (include 3–4 day posting test)

Pre-delivery

- Unit tests for core backend logic.
- Integration tests for API endpoints.
- E2E test checklist executed by developer.

Acceptance tests (run on staging with your test accounts)

1. Add Instagram account and YouTube account; upload YouTube JSON and verify token expiry metadata displayed.
2. Create a connection (Insta → YouTube) with sample source; run manual "Run now" → job completes, video forwarded and appears in destination.
3. Verify automatic forwarding: post new reel on source account and confirm it is forwarded automatically.
4. Upload proxy and associate with connection; verify job uses proxy and health check passes.
5. Force failures:
    - Expire a token (simulate) → Telegram alert fired with correct details.
    - Simulate captcha → job paused and Telegram alert with details; manual mark-as-solved → job resumes.

    - Cause network/proxy failures → retries happen and on success, success alert posted.

6. Analytics: verify daily/weekly numbers for forwarded jobs and success/failure appear correctly.
7. Test video upscaling & green-screen toggles for a connection and verify resulting video changes.
8. Test 3–4 day continuous posting test:
    - Must run for at least 72 hours with automated forwards from multiple connections (3–5) and show stability metrics. Developer to present logs and metrics at end.

Acceptance is when all above work with your test accounts and dev provides fixes for any critical path failures found during 3–4 day test.

---

16. Developer tasks & expectations

- Review existing Python scripts and repo; propose minimal refactor plan or full rewrite if required.
- Provide architecture diagram and justify chosen tech stack.
- Implement APIs and frontend per requirements; implement Celery/Redis worker and Docker compose.
- Implement Telegram notification integration (bot must post to channel; messages must include required details).
- Implement analytics and dashboard visualizations.
- Implement logging, retry/backoff, dead-letter queue and captcha handling flow.
- Provide deployment guide, test plan, and run acceptance tests with you.
- Provide 3–4 day post-deployment support window.
- OAuth flow in UI for YouTube and Facebook (automatic re-auth).
- Automated captcha solving integration (pay-per-solve) as optional plugin.

---

17. Optional / Nice-to-have (prioritize after MVP)

- SaaS-ready multi-tenant design (RBAC, tenant isolation).
- Support for S3 storage.
- Rate-limit & scheduling policies to avoid platform limits.

---

18. Timeline & milestones (developer to propose)

- **Milestone 1**: Project kickoff, repo review, architecture + tech-stack sign-off.

- **Milestone 2**: Core backend API + DB schema + token handling.
- **Milestone 3**: Worker queue, job orchestration (start/stop/retry), Telegram alerts.
- **Milestone 4**: Frontend basic (login, accounts, connections, run controls).
- **Milestone 5**: Analytics dashboard, logs, proxy manager.
- **Milestone 6**: Acceptance testing & 3–4 day posting test.
- **Milestone 7**: Handover, docs, post-deployment support.
