## Identity & Sync Architecture – GDPR-Compliant Profile Backup & Model Delivery

### Context and Goals

The Wellness at Work (WaW) desktop MVP is an offline application (PyQt UI with embedded ML) managing user profiles locally. By Q3 2025, the goals include:

* Enabling secure cloud backup of user profiles.
* Providing automatic remote ML model updates.
* Ensuring strict GDPR compliance (data privacy, opt-in consent, right to erasure).

The solution emphasizes simplicity, incremental deployment, and minimal engineering complexity.

Non-Goals:

* UI or local ML redesign.
* Large-scale, multi-tenant cloud infrastructure.
* User authentication beyond implicit profile identity.

### Component Diagram

Client-side components securely manage profile storage and sync with cloud services for backups and model updates. (See architect\_diag.png)

### Client-Side Components

* User Interface (UI): Desktop PyQt-based UI managing user interactions with profiles.
* Identity Service: Handles local profile CRUD operations via AES-256 encrypted database (profile.db, SQLite/SQLCipher).
* Sync Service: Background service syncing profiles and fetching ML model updates, periodically checking for changes and updates.

### Cloud-Side Components

* Cloud Sync API: REST service (FastAPI/Flask) managing profile uploads and model distribution via HTTPS.
* Profile Backup Store: Encrypted storage of profile data, initially a simple DB scalable to blob storage.
* Model Repository: Stores and serves the latest ML model (model.bin).

### Component Interaction

* UI ↔ Identity Service: Local gRPC communication (strongly typed, efficient).
* Identity Service ↔ profile.db: Secure local data persistence.
* Sync Service → Cloud API: HTTPS REST communication for profile uploads and model downloads.
* Cloud API ↔ Profile/Model Storage: Encrypted data storage and transmission.

### Data Flow

Profile Backup Flow:

1. User updates profile via UI (gRPC: UpdateProfile).
2. Identity Service persists changes to encrypted profile.db.
3. Sync Service detects profile changes (via polling/file watcher).
4. Sync Service uploads profile JSON via HTTPS POST to Cloud Sync API.
5. Cloud securely updates the Profile Backup Store.
6. API confirms; sync timestamp updated locally.
7. Profile securely backed up, enabling recovery/sync across devices.

ML Model Update Flow:

1. Sync Service queries Cloud API regularly (GET /model/latest).
2. If available, downloads new model.bin via HTTPS.
3. Performs integrity check (SHA-256); discards if failed and retries later.
4. Local service loads new model at the next opportunity.
5. Previous stable model retained for rollback if necessary.

### Key Design Decisions

* Protocols:
  + Local: gRPC (clarity, efficiency).
  + Cloud: HTTPS REST (simplicity, compatibility).
* Encryption:
  + At-rest: SQLCipher/AES-256.
  + In-transit: TLS HTTPS.
* Reliability: Background sync ensures resilience, automatic retries, and graceful failure handling.
* Versioning: Models and profiles versioned to ensure compatibility and easy rollbacks.

### Privacy and GDPR Compliance

* Explicit Consent: Backup strictly opt-in with transparent user choice.
* Right to Erasure: Easy deletion of profile data locally and in the cloud.
* Minimalism: Only essential data synced; unnecessary logs or metadata avoided.
* Security: Data encrypted at rest and in transit; audit trails maintained for transparency.

### Phase-in Implementation Plan

1. Local Refactoring:
   * Modularize Identity Service and encrypt local storage (SQLCipher).
   * Expose clear gRPC methods for profile management.
2. Sync Service Development:
   * Implement background process for cloud synchronization and model fetching.
   * Initial development using a mock Cloud API.
3. Cloud API Implementation:
   * Minimal REST API endpoints (POST /profile, GET /model/latest).
   * Simple storage backend initially, designed for scalability.
4. Integration & Testing:
   * Comprehensive end-to-end tests including offline handling, model updates, and profile synchronization.
   * Validate robustness thoroughly prior to public release.
5. Gradual Rollout:
   * Disabled initially for public users; enabled gradually starting with beta/internal testing.
   * Progressive activation reduces risks and potential impacts.
6. Backward Compatibility:
   * Older application versions remain functional.
   * Clear API versioning strategy established.

### Design Rationale and Alternatives

We chose a REST-based Sync Service for the MTP-stage POC to validate core backup and model-update flows with minimal infrastructure and engineering overhead. Alternative designs considered included:

* gRPC Mesh: Higher performance but increased setup complexity.
* OAuth-based Auth: Secure but out of scope for initial MVP.
* Multi-tenant Microservices: Greater scalability yet too heavy for POC.

This lean approach accelerates market-fit learning and reduces time-to-POC.

### Future Enhancements

We deliberately chose a simple, REST-based Sync Service design because we’re in an MTP-stage POC phase: it lets us validate core backup and model-update flows with minimal infrastructure and engineering overhead. More elaborate options—like a full gRPC mesh, multi-tenant microservices, or OAuth-based auth—would add complexity and delay market-fit learning.

Enhancement Roadmap (Phased Integration):

1. Performance & Footprint (Phase 1–2):
   * Rewrite client sync loops in Rust modules alongside existing Python Sync Service to reduce memory use and boost throughput.
2. ML Integration (Phase 2–3):
   * Embed a lightweight reinforcement-learning agent within the Sync Service to prioritize profiles or model deltas. Initially prototype as an optional feature flag to compare against baseline.
3. Sync Optimization (Phase 3):
   * Transition from periodic polling to event-driven or delta-driven syncing (WebHooks or gRPC streams). Implement alongside REST endpoints, toggleable via configuration.
4. Option A: S3 Intelligent-Tiering + CDN
   * Phase 4: Migrate profile snapshots and model binaries to Intelligent-Tiering buckets. Configure CloudFront distribution to deliver models, running in parallel with the existing storage mock for validation.
5. Option B: MongoDB Unstructured-Data + RAG-Driven Sync
   * Phase 4–5: Stand up a MongoDB test cluster. Mirror existing profile store into MongoDB documents, enable Change Streams, and validate RAG-based incremental sync against REST baseline.
6. Scalable Cloud Storage & Data Sync (Combined):
   * After validating both options, select or hybridize storage and sync strategies. Implement final data migration scripts and retire legacy file-based backup.

This phased plan allows new modules—Rust, RL agent, event-based sync, S3/CDN, MongoDB—to be integrated alongside current services, controlled via feature flags and incremental rollout. Each phase includes A/B validation against the REST-based POC to ensure performance gains and operational stability.

### Conclusion and Wayforward

The document now aligns the visual and textual architecture, explains our design choices, and outlines phased enhancements. Next steps include comprehensive integration testing, validating GDPR compliance, and rolling out incremental features guided by real user feedback to build a robust, scalable, and cost-effective solution.