ChainLynx Bikepacking App – Backend Architecture Options

Pre-Release Technical Documentation — November 2025

1. Overview

This document evaluates backend architecture options for the ChainLynx Bikepacking App. Each option is assessed for scalability, privacy, data persistence, and implementation complexity. The goal is to identify an architecture that best supports offline-first operation, user data sovereignty, and community scalability.

2. Option 1 — Federated Model (ActivityPub)

Concept: A fully decentralized backend built on the ActivityPub protocol (used by Mastodon). Trail reports would be represented as custom Activity objects distributed across multiple servers.

Pros: • Maximum user control and data ownership.
• Censorship-resistant architecture with distributed hosting.

Cons: • Complex implementation requiring new Activity types for GPS and structured data.
• Aggregating trail reports across instances is unreliable and slow.
• Data persistence cannot be guaranteed if smaller servers go offline.

Recommendation: Not suitable for ChainLynx. While it offers strong decentralization, reliability and scalability limitations make it impractical for structured route data management.

3. Option 2 — Hybrid, Privacy-First Model (Recommended)

Concept: This model separates private user data from shared community data. Personal journals and preferences remain in user-owned cloud storage, while community data is stored in a lightweight backend.

Data Separation: • Personal Data — Stored in the user's cloud account (iCloud, Google Drive, OneDrive).
• Community Data — Trail reports, photos, and meetups managed by a minimal backend for persistence.

Pros: • Ensures persistence of community data even after account deletion.
• Scalable, familiar architecture for developers.
• Privacy by design — the backend stores minimal personally identifiable information (PII).

Cons: Requires transparent trust and clear privacy communication to maintain user confidence.

Recommendation: The hybrid model is the most practical and privacy-aligned solution for ChainLynx. It balances user control, reliability, and technical feasibility. Future backend development should assume this architecture.

4. Privacy-Respecting Content Storage Strategy

A. Text and Metadata: Use PostgreSQL with PostGIS for efficient geospatial queries (e.g., "find all reports in current map view"). Store only essential fields like content, coordinates, and a hashed author ID. Avoid emails or direct identifiers.

B. Media Storage: Use an S3-compatible object storage solution for images and videos — never store media directly in the database. Recommended providers include Backblaze B2 or Wasabi. Self-hosted alternative: MinIO.

Upload Workflow: 1. The app requests upload permission from the backend.
2. Backend generates a secure, time-limited pre-signed URL.
3. The app uploads directly to storage.
4. Backend records the file URL and metadata in the database.

5. Data Persistence and Account Deletion

Each backend account stores only a hashed ID, username, and avatar URL — no passwords or emails. When a user deletes their account: • The user record is permanently removed.
• All related posts are anonymized by clearing the author hash.
Result: Trail reports remain public but no longer linked to personal identity, preserving community value while maintaining privacy.

6. Integration with React Native

Implementation: Integration is straightforward using REST or GraphQL APIs.
Authentication Flow: 1. User signs in via Google or Apple using SDKs (e.g., @react-native-google-signin/google-signin).
2. The app receives an ID token (JWT).
3. The app includes this token in Authorization headers.
4. Backend verifies the token’s signature using provider public keys and extracts the user’s unique sub ID.
5. The sub ID is hashed for backend operations, enabling secure, stateless authentication without passwords.

This approach achieves privacy-preserving authentication and a smooth developer workflow consistent with open-source best practices.