ChainLynx Bikepacking App - Phase 1: Project Scaffolding & Core Architecture

Pre-Release Technical Documentation - November 2025

1. Overview
Phase 1 establishes the foundational architecture for the ChainLynx Bikepacking App, defining technical structure, codebase organization, and collaboration workflow. The goal is to deliver a functioning scaffolding that supports core features such as offline operation, authentication, and modular integration.

2. Objectives
Primary Goals:
• Set up a maintainable codebase for cross-platform development.
• Establish backend and database connections for community data.
• Implement offline storage for user data and maps.
• Create developer standards for contributions and testing.

3. Core Technologies
Frontend: React Native with TypeScript, using Expo for deployment and NativeWind (Tailwind for React Native) for styling.
Backend: Node.js (Fastify) with PostgreSQL + PostGIS for geospatial data.
Storage: Cloud-based (user-owned) options like iCloud, Google Drive, or OneDrive for personal data; PostgreSQL for community data.
AI Runtime: TensorFlow Lite or ONNX Runtime for on-device inference.
Version Control: GitHub repository under the organization "chainlynx-app".

4. Project Setup
Setup Steps:
1. Clone the repository from GitHub.
2. Install dependencies via npm or yarn.
3. Configure environment variables for API keys, map tiles, and backend URLs.
4. Launch the development environment using Expo.
5. Run backend services locally with Docker Compose for PostgreSQL and Node.js.

Tools:
• Expo CLI for mobile builds.
• VS Code as the primary IDE.
• Docker for backend containerization.
• Postman for API testing.
• GitHub Actions for continuous integration (planned).

5. Repository Structure
A clean and modular structure ensures scalability and ease of maintenance.
/chainlynx-bikepacking
■■■ /app - React Native app source code
■ ■■■ /components - Shared UI components

```
■ ■■■ /screens - Feature screens (Feed, Map, Profile)
■ ■■■ /context - Context API for state management
■ ■■■ /hooks - Custom React hooks
■ ■■■ /assets - Icons, fonts, and images
■ ■■■ /utils - Shared helpers and constants
■■■ /server - Node.js backend
■ ■■■ /routes - REST API endpoints
■ ■■■ /controllers - Request handling logic
■ ■■■ /models - PostgreSQL ORM (Prisma or Sequelize)
■ ■■■ /services - Background tasks and integrations
■■■ /docs - Project documentation (PDFs, manifests)
```

6. Development Workflow

Branching Strategy: Follows GitHub Flow.
• main - stable release branch.
• dev - integration branch for ongoing work.
• feature/* - branches for new features.
• fix/* - branches for bug fixes.
Pull requests are required for all merges.

Issue Management:
All work is tracked in GitHub Issues, labeled by category (Feature, Bug, Enhancement). Milestones correspond to development phases, with each assigned to a sprint cycle (2-3 weeks).

7. Contributor Roles

The ChainLynx project invites contributors from multiple domains:
• **UI/Frontend Developers:** Build React Native components, implement layouts, and ensure visual consistency using NativeWind. Handle both interface design and development for a cohesive user experience.
• **Backend Developers:** Build Node.js endpoints, database schemas, and APIs for managing community data and authentication.
• **AI Developers:** Optimize offline inference models, train small-scale domain models, and ensure compatibility with mobile runtimes.
• **Community Contributors:** Provide testing, documentation updates, feedback, and assist in data validation and content moderation.

8. Milestones

Phase 1 Goals:
1. Establish repository and code scaffolding.
2. Implement authentication (Google and Apple Sign-In).

3. Build basic map view and offline storage.
4. Integrate simple feed prototype and local caching.
5. Publish developer documentation for contributors.

Future milestones (Phase 2-3) will focus on messaging integration, AI assistants, and backend scalability improvements.

9. Recommendation

Phase 1 provides the foundation upon which all subsequent ChainLynx development will build. Maintaining consistent documentation, commit hygiene, and modular structure from the start ensures scalability and a smooth onboarding experience for future contributors.