GatherUp — Phase-by-Phase Detailed Plan (Phases 1 to 7)

This document contains full detailed explanations for Phases 1 through 7 including goals, exact files to implement (backend, frontend, shared), and acceptance checklists. Use these sections to implement, test, and verify each phase.

Phase 1 — Auth: Register / Login / Refresh (MVP) Goal: Users can register, login, receive access + refresh tokens; Android stores tokens and uses them for authenticated requests.

Backend Files:
- backend/go/api/routes.go
- backend/go/api/handlers_auth.go
- backend/go/api/handlers_user.go
- backend/go/api/middleware.go
- backend/go/auth/jwt.go
- backend/go/auth/tokens.go
- backend/go/auth/passwd_helpers.go
- backend/go/models/user.go
- backend/go/repository/user_repo.go
- backend/go/service/auth_service.go
- backend/go/db/conn.go
- backend/go/db/migrations/0001_auth_schema.sql
- backend/go/config/config.go

Backend Responsibilities and Notes (detailed):
- Implement secure password hashing using bcrypt or Argon2, with salts and cost parameters in config.
- JWT creation: include claims sub (user id), exp, iat, roles, and a key id (kid) if rotating keys. Sign with HMAC or RSA based on config.
- Store refresh tokens server-side with hashed token representation (rotate on each use) and associated expiry and device metadata.
- Implement middleware to validate access token on protected routes; refresh token endpoint must validate and issue a new access token (and rotate refresh token).
- Provide /auth/register, /auth/login, /auth/refresh endpoints and a protected /api/me endpoint for verification.
- Add input validation and rate-limiting on auth endpoints.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/di/HiltApp.kt
- mobile/android/app/src/main/java/com/gatherup/di/NetworkModule.kt
- mobile/android/app/src/main/java/com/gatherup/security/TokenStore.kt
- mobile/android/app/src/main/java/com/gatherup/data/api/AuthService.kt
- mobile/android/app/src/main/java/com/gatherup/data/repository/AuthRepositoryImpl.kt
- mobile/android/app/src/main/java/com/gatherup/vm/AuthViewModel.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/auth/LoginScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/auth/RegisterScreen.kt
- mobile/android/app/src/main/java/com/gatherup/data/models/User.kt
- mobile/android/app/src/main/java/com/gatherup/data/api/ApiResponse.kt

Frontend Responsibilities and Notes (detailed):
- NetworkModule: provide Retrofit, OkHttpClient with an AuthInterceptor that attaches Bearer token and handles 401 by triggering token refresh flow.
- TokenStore: use EncryptedSharedPreferences or Android Keystore to persist access and refresh tokens securely; expose suspend functions to read/write tokens.

- AuthRepository: wrap network calls and coordinate TokenStore; handle refresh token rotation seamlessly.
- AuthViewModel: provide UI state, handle login/register flows, expose errors and loading states to Compose screens.
- Compose Screens: LoginScreen and RegisterScreen should validate inputs, show progress, and call ViewModel actions. On success navigate to main flow.

Shared / Contracts:
- backend/go/proto/api.proto (define auth messages)
- backend/go/api/openapi.yaml (or openapi_v1.yaml snippet for auth endpoints)
- contracts/auth.md (recommended: JWT claims, expiry durations, rotation rules)

Acceptance checklist (Phase 1):
- Register endpoint: POST /auth/register creates user, passwords stored hashed.
- Login endpoint: POST /auth/login returns { access_token, refresh_token, expires_in }.
- Refresh endpoint: POST /auth/refresh returns new access token and rotates refresh token.
- Protected endpoint /api/me requires Authorization header and returns user info.
- Android: Login and Register screens working; TokenStore securely stores tokens; AuthInterceptor attaches Authorization header.
- Integration: Android can call /auth/login, receive tokens, and call protected /api/me successfully.

Phase 2 — Basic Feed (Posts) & User Profile Goal: Create/read posts and view user profiles (REST). No websockets yet.

Backend Files:
- backend/go/api/handlers_posts.go
- backend/go/api/handlers_user.go (extended)
- backend/go/models/post.go
- backend/go/repository/post_repo.go
- backend/go/service/post_service.go
- backend/go/db/migrations/0002_posts_schema.sql
- backend/go/api/routes.go (updated)

Backend Responsibilities and Notes:
- Implement post creation, listing with pagination, retrieval by id, and basic permissions (author-only edits/deletes).
- Use cursor-based pagination or page+limit; decide and keep consistent with frontend. Provide default limit and max allowed limit.
- Ensure posts have fields: id, author_id, content, media_urls, created_at, updated_at, deleted_at (soft delete).
- Index important columns for performance (created_at, author_id).

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/data/api/PostService.kt
- mobile/android/app/src/main/java/com/gatherup/data/repository/PostRepositoryImpl.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/FeedScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/PostDetailScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/components/posts/PostCard.kt
- mobile/android/app/src/main/java/com/gatherup/vm/FeedViewModel.kt

Frontend Responsibilities and Notes:
- Implement FeedViewModel to call PostRepository and expose paged data to Compose UI.
- PostCard component should display author avatar, content, media gallery snippet, and actions (like,

comment, share).
- PostRepository should map API responses to domain models and handle paging.

Shared / Contracts:
- contracts/openapi_v1.yaml (posts section added)

Acceptance checklist (Phase 2):
- Backend: create post, list posts (paginated), get post by id.
- Android: feed loads posts, opens post detail, uses Auth header correctly.

Phase 3 — Real-Time Chat (WebSocket MVP) Goal: One-to-one chat via WebSocket with persistent storage for messages.

Backend Files:
- backend/go/ws/hub.go
- backend/go/ws/connection.go
- backend/go/ws/router.go
- backend/go/ws/handlers.go
- backend/go/ws/redis_pubsub.go (optional for scaling)
- backend/go/models/message.go
- backend/go/repository/message_repo.go
- backend/go/service/chat_service.go

Backend Responsibilities and Notes:
- Implement WebSocket handshake that accepts token (e.g., query param ?token= or Authorization header during upgrade).
- Authenticate token, map to user session, and store mapping from userID to active connection(s).
- Hub handles broadcasting messages to specific user connections and to rooms (if supporting group chats).
- Persist messages with read status and timestamps. Provide REST endpoints to fetch chat history for a conversation.
- If multiple backend instances exist, use redis_pubsub.go to fan-out messages between instances.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/ws/WsClient.kt
- mobile/android/app/src/main/java/com/gatherup/ws/WsMessageHandler.kt
- mobile/android/app/src/main/java/com/gatherup/ws/WsModels.kt
- mobile/android/app/src/main/java/com/gatherup/ws/WsReconnectionManager.kt
- mobile/android/app/src/main/java/com/gatherup/ws/WsHeartbeatManager.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/ChatListScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/ChatThreadScreen.kt
- mobile/android/app/src/main/java/com/gatherup/vm/ChatViewModel.kt

Frontend Responsibilities and Notes:
- WsClient: connect using OkHttp WebSocket or recommended client, send/receive JSON envelope messages.
- WsReconnectionManager: exponential backoff, jitter, and state exposed via StateFlow to UI.
- WsHeartbeatManager: send periodic pings or no-op messages to keep connection alive.
- ChatViewModel: maintain messages in a local cache (Room) and update UI as incoming messages arrive.

Shared / Contracts:
- backend/go/proto/message.proto (defines WS envelope and message types)
- contracts/ws_message_schema.json (sample messages and event types)

Acceptance checklist (Phase 3):
- Backend: WebSocket auth on handshake, route messages, persist messages reliably.
- Android: WsClient connects (wss), sends and receives messages, messages display in UI and persist locally.

Phase 4 — Tournaments (CRUD + Leaderboard Worker) Goal: Users can create/join tournaments; backend worker updates leaderboards.

Backend Files:
- backend/go/api/handlers_tournaments.go
- backend/go/models/tournament.go
- backend/go/repository/tournament_repo.go
- backend/go/service/tournament_service.go
- backend/go/db/migrations/0003_tournaments.sql
- backend/go/worker/leaderboard.go

Backend Responsibilities and Notes:
- Tournament model: id, name, type, participants, matches, status, start_time, end_time, metadata.
- Worker computes leaderboard periodically (cron / scheduled job) based on match results and scoring rules.
- Provide endpoints to create tournament, join, leave, update matches, and fetch leaderboard.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/data/api/TournamentService.kt
- mobile/android/app/src/main/java/com/gatherup/data/repository/TournamentRepositoryImpl.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/TournamentListScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/TournamentDetailScreen.kt
- mobile/android/app/src/main/java/com/gatherup/ui/screens/CreateTournamentScreen.kt
- mobile/android/app/src/main/java/com/gatherup/vm/TournamentViewModel.kt

Acceptance checklist (Phase 4):
- Backend: create and join tournaments, worker computes leaderboard on schedule.
- Android: list tournaments, join/leave UI, view leaderboard.

Phase 5 — Notifications & Background Jobs (Push + In-App) Goal: Real-time in-app notifications via WS + push notifications via worker.

Backend Files:
- backend/go/models/notification.go
- backend/go/repository/notification_repo.go
- backend/go/service/notification_service.go
- backend/go/worker/notifier.go
- backend/go/ws/hub.go (updated to fan-out notifications)
- backend/go/ws/handlers.go (updated)

Backend Responsibilities and Notes:
- Notification model: id, user_id, type, payload, read, created_at.
- Worker picks up queued notification jobs and sends push via FCM/APNS; also writes to DB and instructs

ws hub to deliver in-app notifications.
- Provide APIs for clients to fetch and mark notifications as read; support pagination.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/notifications/NotificationManager.kt
- mobile/android/app/src/main/java/com/gatherup/ui/components/NotificationCenter.kt
- mobile/android/app/src/main/java/com/gatherup/work/SyncWorker.kt

Frontend Responsibilities and Notes:
- NotificationManager handles registering for push tokens, sending token to backend, and preparing channel/on-device behavior.
- NotificationCenter shows recent notifications and allows marking them read. Use WorkManager for background sync tasks.

Acceptance checklist (Phase 5):
- Backend: queue notifications, worker sends push notifications; ws in-app notifications delivered real-time.
- Android: push notifications appear when app is backgrounded; in-app notification center shows items and updates read status.

Phase 6 — Media Uploads & CDN (R2) + Offline sync Goal: Allow media uploads (images/videos) via presigned URLs and offline sync for posts.

Backend Files:
- backend/go/storage/r2_client.go
- backend/go/storage/upload_handler.go
- backend/go/storage/file_utils.go

Backend Responsibilities and Notes:
- Implement presigned URL generation for uploads; validate content type and size limits.
- Provide responses including upload_url, method (PUT), required headers, and expiry.
- Optionally provide chunked/resumable upload endpoints for large files.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/ui/components/posts/MediaGallery.kt
- mobile/android/app/src/main/java/com/gatherup/data/uploads/UploadManager.kt
- mobile/android/app/src/main/java/com/gatherup/data/paging/PostRemoteMediator.kt

Frontend Responsibilities and Notes:
- UploadManager uses WorkManager for background uploads and retries; supports resumable uploads or single PUT to presigned URL.
- PostRemoteMediator handles synchronizing local DB with server pages and marks locally created posts as pending until uploaded & acknowledged.

Acceptance checklist (Phase 6):
- Backend: presign URL endpoint returns signed PUT URL and upload headers.
- Android: user can attach media, WorkManager uploads using presigned URL, feed shows uploaded media after sync.

Phase 7 — Observability, CI, Security hardening, Tests Goal: Add logs, metrics, security checks, CI pipelines, and

comprehensive tests.

Backend Files:
- backend/go/pkg/logger/logger.go
- backend/go/pkg/metrics/metrics.go
- backend/go/.github/workflows/ci.yml

Backend Responsibilities and Notes:
- Structured logging, correlation IDs, and tracing hooks. Export metrics for Prometheus.
- Unit and integration tests for services and handlers; integration tests to run against a test DB in CI.

Frontend Files:
- mobile/android/app/src/main/java/com/gatherup/observability/CrashReporter.kt
- mobile/android/app/src/main/java/com/gatherup/observability/Analytics.kt
- mobile/android/.github/workflows/android-ci.yml
- mobile/android/detekt.yml

Frontend Responsibilities and Notes:
- Add unit tests, Compose UI tests, and instrumentation tests that run in CI.
- Integrate crash reporting and analytics with privacy-conscious settings.

Acceptance checklist (Phase 7):
- CI runs: linters, unit tests, backend integration tests.
- Logging with structured logs and traces; metrics exportable (Prometheus).
- Security review: TLS everywhere, rate limiting, OWASP checks.

Cross-phase: Contracts & Versioning (always required)
Keep authoritative contracts under contracts/ or backend/go/proto/ with versioning.
Files to maintain for contracts:
- contracts/api_v1.proto or backend/go/proto/api.proto
- contracts/openapi_v1.yaml or backend/go/api/openapi.yaml
- contracts/ws_message_schema.json
- contracts/auth.md

Rule: Any breaking change requires cross-team PR and version bump.

Quick-start recommendation (what to code first)
1. Phase 1 (Auth) — implement backend handlers + migrations + JWT; implement Android NetworkModule, TokenStore, AuthService, Login/Register screens.
2. Add contracts/api_v1.proto with the auth messages and commit it — frontend uses it to generate models.
3. Verify end-to-end: Android login → protected /api/me.
4. Proceed Phase-by-Phase, merging only non-breaking contract changes after both teams agree.

End of document.