SocialSpot

Developer Guide

A comprehensive technical reference for building, extending, and maintaining the SocialSpot iOS application.

Platform: iOS (iPhone and iPad) | Language: Swift 5.0 | IDE: Xcode 16.4+

`Bundle Identifier: com.kent.SocialSpot`

`Author: Ben Korjenek`

Build Environment

This section outlines the tools, configurations, and processes required to compile and run the SocialSpot project.

Item	Value
Language	Swift 5.0
IDE	Xcode 16.4+
iOS Deployment Target	18.5
Platform	iOS (iPhone and iPad)
Bundle Identifier	com.kent.SocialSpot

The project is a native SwiftUI application. There is no cross-platform tooling (React Native, Flutter, etc.). All UI is built with SwiftUI and MapKit.

Compiler and Toolchain

Xcode provides the Swift compiler, linker, and all Apple SDK frameworks. No additional compiler installation is required beyond a standard Xcode setup. The project targets arm64 for both device and simulator.

Build Management

The project uses Xcode's native build system (xcodeproj). There is no Makefile, CMake, or custom build script. Building can be performed through the following methods:

- **Xcode GUI:** Open SocialSpot.xcodeproj, select the SocialSpot scheme, and press Cmd+R.
- **Command line:** Use the xcodebuild command with the appropriate scheme and simulator destination.

```
xcodebuild -project SocialSpot/SocialSpot.xcodeproj \
  -scheme SocialSpot \
  -destination 'platform=iOS Simulator,name=iPhone 17 Pro' \
  build
```

Use xcrun simctl list devices to see available simulator names on your machine.

Automated Builds / CI

There is currently no CI/CD pipeline configured. No GitHub Actions workflows, Fastlane setup, or other automated build systems exist. Builds and tests are run manually from Xcode or the command line.

Dependencies

Dependencies are managed through Swift Package Manager (SPM), integrated directly into the Xcode project. There is no CocoaPods Podfile or Carthage Cartfile.

Direct Dependency

Package	Version	Repository
supabase-swift	2.41.0 (min 2.5.1)	github.com/supabase/supabase-swift

The following Supabase modules are imported across the project:

- **Auth** — user authentication (sign up, sign in, session management)
- **PostgREST** — database queries (profiles, links, user_locations, user_settings)
- **Storage** — profile image upload/download
- **Realtime** — real-time location channel subscriptions

Transitive Dependencies

These are pulled in automatically by supabase-swift and do not need to be managed directly:

Package	Version
swift-concurrency-extras	1.3.2
swift-http-types	1.5.1
swift-crypto	4.2.0
swift-asn1	1.5.1
swift-clocks	1.0.6
xctest-dynamic-overlay	1.8.1

Where Dependencies Are Listed

- **Package reference:** SocialSpot.xcodeproj/project.pbxproj (the XCRemoteSwiftPackageReference section)
- **Resolved versions:** SocialSpot.xcodeproj/project.xcworkspace/xcshareddata/swiftpm/Package.resolved

To update dependencies, open the project in Xcode and navigate to File > Packages > Update to Latest Package Versions.

Secrets and Configuration

The Supabase API key is stored in an xcconfig file that is git-ignored:

- **Template:** SocialSpot/Secrets.template.xcconfig — committed to the repo, contains an empty placeholder.
- **Active file:** SocialSpot/Secrets.xcconfig — not committed, must be created locally by each developer.

To set up secrets:

1. Copy Secrets.template.xcconfig to Secrets.xcconfig in the same directory.
2. Fill in your Supabase anonymous key:

```
supabaseKey = your-supabase-anon-key-here
```

3. The key is read at runtime via Info.plist which references $(supabaseKey).

The Supabase project URL is hardcoded in SupabaseConfig.swift at the SupabaseManager singleton.

Project Structure

All application source files live under SocialSpot/SocialSpot/. The directory layout is as follows:

File	Purpose
SocialSpotApp.swift	App entry point, root navigation, auth routing
ContentView.swift	Main map screen, location manager, nearby user polling
SupabaseConfig.swift	SupabaseManager, AuthService, UserService, LocationService
Models.swift	User, SocialMediaAccount, SocialMediaPlatform data models
NearbyUsersList.swift	Bottom sheet list of nearby users
UserProfileCard.swift	Full profile card shown when tapping a user
AuthViews.swift	Welcome, sign in, and sign up screens
OnboardingViews.swift	Post-signup onboarding flow (name, birthday, socials)
EditProfileView.swift	Edit profile form
ProfileSettingsView.swift	Settings sheet (profile display, location toggle, sign out)
CameraPicker.swift	UIImagePickerController wrapper for camera/photos
ImageURLResolver.swift	Resolves profile image strings to displayable URLs
FullScreenImageView.swift	Full screen image viewer
Info.plist	App permissions and configuration
Secrets.template.xcconfig	Secrets template
Assets.xcassets	App icons and asset catalog

Architecture Overview

The app follows a straightforward SwiftUI + ObservableObject pattern. The core service classes are:

- **AuthService** (@MainActor, ObservableObject) — manages authentication state, injected as an EnvironmentObject from the app root.
- **UserService** (@MainActor, ObservableObject) — handles profile updates, social account updates, and account deletion.
- **LocationService** (@MainActor, ObservableObject) — handles location upserts to Supabase and fetches nearby users.
- **LocationManager** (NSObject, CLLocationManagerDelegate, ObservableObject) — wraps CoreLocation for continuous GPS updates.

Navigation Flow

SocialSpotApp creates the AuthService, then RootView checks auth state:

- Not authenticated → WelcomeView → SignInView / SignUpView
- Authenticated, no birthday → OnboardingFlow
- Authenticated, complete profile → ContentView (map + nearby list)

Backend (Supabase)

The app communicates with a Supabase PostgreSQL backend. The relevant tables are:

Table	Purpose
profiles	User profile data (display_name, bio, school, birthday, profile_image_url)
links	Social media usernames (instagram, snapchat, tiktok, twitter, linkedin, youtube, other)
user_locations	GPS coordinates (user_id, lat, lng, updated_at)
user_settings	Preferences (location_enabled, profile_public, location_ttl_minutes)

Row Level Security (RLS) is enabled. Key policies:

- All users can read user_locations (public SELECT).
- Users can only write their own rows (INSERT/UPDATE policies check auth.uid() = user_id).
- The user_id column stores UUIDs in lowercase format to match auth.uid().

How to Make Changes

Adding a New View

1. Create a new .swift file in SocialSpot/SocialSpot/.
2. Define a struct conforming to View.
3. If the view needs auth state, add @EnvironmentObject var authService: AuthService.
4. Navigate to the new view from an existing view using NavigationLink or sheet.
5. The Xcode project uses synchronized file groups, so new files in the directory are automatically included in the build.

Adding a New Supabase Table/Query

1. Define a Codable struct matching the table's column names (use CodingKeys for snake_case mapping).
2. Add query methods to the appropriate service class in SupabaseConfig.swift.
3. Use .lowercased() on any user_id values to match Supabase's auth.uid() format.
4. Ensure the table has appropriate RLS policies in the Supabase dashboard.

Adding a New Swift Package Dependency

In Xcode, go to File > Add Package Dependencies. Enter the repository URL and select a version rule. The dependency will be added to the xcodeproj and resolved in Package.resolved.

Code Style Expectations

Naming Conventions

- **PascalCase** for types, structs, classes, enums, and protocols (e.g., UserProfileCard, AuthService).
- **camelCase** for properties, methods, and local variables (e.g., currentUser, fetchNearbyUsers).
- **snake_case** only in Encodable/Decodable structs that map directly to Supabase column names (e.g., user_id, display_name). Use CodingKeys to bridge between Swift camelCase and database snake_case.

SwiftUI Patterns

- Use @State private var for local view state.
- Use @StateObject for owned ObservableObject instances created by the view.
- Use @EnvironmentObject for shared objects passed down the view hierarchy.
- Use @AppStorage for simple persistent preferences (e.g., satelliteMap, darkModeEnabled).
- Avoid the Combine framework for new code. Use Swift's async/await instead.

Formatting

- 4-space indentation.
- One blank line between methods.
- Use // MARK: - comments to separate logical sections within a file.
- Keep views focused. Extract subviews into separate structs when a body property gets long.

General Guidelines

- Use let for constants, var only when mutation is needed.
- Avoid force unwrapping (!). Use guard let or if let for optionals.
- Keep error handling with do/catch around Supabase calls. Print errors for debugging but do not crash.
- Favor SwiftUI's declarative style. Avoid UIKit wrappers unless necessary.

Automated Testing

Testing Frameworks

- **XCTest** — used for all unit and UI tests.
- **Swift Testing framework** — one placeholder test exists (SocialSpotTests.swift), but XCTest is the primary framework.
- **XCUITest** — used for UI automation tests (app launch, screenshot capture).

Test Case Locations

All test files are organized in the repository under the following structure:

Directory / File	Description
SocialSpotTests/ModelsTests.swift	User model: age calculation, social parsing, distance
SocialSpotTests/ImageURLResolverTests.swift	URL resolution: empty, SF symbol, HTTP cache-buster
SocialSpotTests/NearbyUsersListTests.swift	Sorting by distance, nearby count, nil location
SocialSpotTests/SocialMediaPlatformTests.swift	Icon names, colors, display names
SocialSpotTests/SocialSpotTests.swift	Placeholder test
SocialSpotUITests/SocialSpotUITests.swift	Basic launch test, launch performance measurement
SocialSpotUITests/SocialSpotUITestsLaunchTests.swift	Launch screenshot capture
SocialSpot.xctestplan	Test plan (both targets, parallelizable)

Test Count

Category	Count
Unit tests	12 (across 5 files)
UI tests	3 (across 2 files)
Total	15 test cases

How to Run Tests

From Xcode

1. Open SocialSpot.xcodeproj.

2. Select the SocialSpot scheme.
3. Press Cmd+U or go to Product > Test.

From the Command Line

Run the full test suite:

```
xcodebuild test \
  -project SocialSpot/SocialSpot.xcodeproj \
  -scheme SocialSpot \
  -destination 'platform=iOS Simulator,name=iPhone 17 Pro'
```

Run a specific test class:

```
xcodebuild test ... -only-testing:SocialSpotTests/NearbyUsersListTests
```

Run a single test method:

```
xcodebuild test ... -only-
testing:SocialSpotTests/ImageURLResolverTests/testEmptyStringReturnsNil
```

Test Coverage Summary

Test File	What It Tests
ModelsTests.swift	Age computation from birthday string, social account parsing and @ cleanup, distance calculation between coordinates
ImageURLResolverTests.swift	Empty/whitespace returns nil, SF symbol names return nil, HTTP URLs get a cache-busting t= parameter
NearbyUsersListTests.swift	Users sorted by distance ascending, nearby count filters at 152.4m (500 ft), nil location returns original order
SocialMediaPlatformTests.swift	Correct SF Symbol icon names per platform, hex color values, display name mapping (Twitter shows as “X”)
SocialSpotUITestsLaunchTests.swift	App launches successfully, captures a screenshot of the launch screen

Backlog and Known Issues

GitHub Repository

The project is hosted at:

https://github.com/sawyerke/SocialSpot

Issues and pull requests are tracked on GitHub. Check the Issues tab for the current bug list and feature requests.

Known Issues and Limitations

- **University accounts:** Users are not yet able to create accounts using their university .edu emails to enable the university student filtering feature.
- **Birthday editing disabled:** Users set their birthday during onboarding but cannot change it afterward.

Incomplete Features

- **Social account removal:** Users can add social media accounts but cannot remove individual ones from their profile. (Noted in commit a21eb67.)
- **Birthday editing disabled:** The birthday picker is commented out in EditProfileView.swift. Users set their birthday during onboarding but cannot change it afterward.
- **Connect/Star buttons removed:** UserProfileCard.swift previously had Connect and Star action buttons. These were intentionally removed but may be re-added as future features.

Hardcoded Values

- **Default map center:** ContentView.swift defaults to University of Alabama coordinates (33.2098, -87.5692). This is the initial map position before the user's GPS location is resolved.
- **Profile location label:** ProfileSettingsView.swift displays "Tuscaloosa, AL" as a hardcoded string rather than deriving it from the user's actual location via reverse geocoding.
- **Testing radius:** The nearby user radius is currently set to 500 feet (152.4 meters) for testing purposes. The intended production value is 50 feet (15.24 meters). This value appears in SupabaseConfig.swift, ContentView.swift, and NearbyUsersList.swift.

Code Cleanup

- **Debug print statements:** SupabaseConfig.swift contains multiple print() calls in fetchNearbyUsers that log user IDs and query results. These should be wrapped in #if DEBUG blocks or removed before release.

- **Commented-out legacy code:** Models.swift (lines 133–188) contains old User struct and sample data definitions that are fully commented out and should be removed.

Major Future Considerations

- **Scalability of nearby user queries:** The current approach fetches all rows from user_locations and filters client-side by distance. Consider enabling PostGIS on Supabase and using ST_DWithin for server-side geographic filtering.
- **No CI/CD:** There is no automated build or test pipeline. Setting up GitHub Actions for pull request validation would prevent regressions.
- **Supabase SDK versioning:** The project pins supabase-swift at 2.41.0. Major version updates to the SDK may require API changes. Monitor the supabase-swift releases for breaking changes.
- **iOS deployment target:** The app targets iOS 18.5, which limits the user base to devices running the latest OS. Consider whether a lower deployment target is needed for broader reach.