Swift Style Guide

Our team follows these key style guidelines for Swift development:

Code Formatting

- Use standard Swift camelCase naming conventions
- Indent using 4 spaces (no tabs)
- Opening brace on same line, closing on new line:

```swift
if condition {
    // code
}
```

File Header Comments

Every file should start with a standard header:

```swift
//
//  FileName.swift
//  ProjectName
//
//  Created by [Author Name] on [Date]
//
// [Description of the file's purpose]
// [Current status/TODO items]
// [Request to comment changes with name]
```

Naming Conventions

- Variables and functions start lowercase, use camelCase
    - email
    - password
    - signIn()
- View structs start uppercase, use camelCase, end in "View"
    - LoginView
    - RegistrationView

- ViewModels end in "ViewModel"
  - AuthViewModel
- Names should be descriptive and clear

Comments

- Comments explain current status and needed changes
- Leave name when making changes
- Use consistent comment formatting:

```swift
// Background color
Color.blue
    .ignoresSafeArea()

// Form Fields: Email and Password
VStack(spacing: 24) {
    // code
}
```

Project Structure

Our application files are organized using the following directory structure and conventions:

File Naming Conventions

- Views end with "View" (e.g., LoginView, MapView)
- ViewModels end with "ViewModel" (e.g., AuthViewModel)
- Controllers end with "Controller" (e.g., LandmarkController)
- Managers end with "Manager" (e.g., NotificationManager)
- Utility files describe their function (e.g., RoundedCorner)

Organization Principles

- Components/: Contains reusable UI elements
- Core/: Contains main feature modules (Auth, Profile, etc.)
- Root/: Contains primary app views and navigation
- Model/: Contains data structures and types
- Utilities/: Contains helper functions and extensions
- Clear separation of Views from ViewModels
- Related functionality grouped in subfolders

This structure promotes:

- Modular development
- Clear feature organization
- Separation of concerns
- Easy navigation
- Scalable architecture

Directory layout below

```
Seekr/
├── App/
│   └── SeekerApp.swift (main app configuration)
├── Components/
│   ├── InputView
│   └── SettingsRowView
├── Core/
│   ├── Authentication/
│   │   ├── View/
│   │   │   ├── AuthenticationView
│   │   │   ├── LoginView
│   │   │   ├── NotificationSettingsView
│   │   │   └── RegistrationView
│   │   └── ViewModel/
│   │       └── AuthViewModel
│   ├── Notifications/
│   │   └── NotificationManager
│   └── Profile/
│       └── ProfileView
├── Root/
│   ├── MapUtils/
│   │   ├── Compass
│   │   ├── HideMapOverlay
│   │   ├── LandmarkController
│   │   └── LocationManager
│   ├── ContentView
│   ├── EndRouteMenu
│   ├── MainView
│   ├── MapView
│   ├── PinsView
│   ├── SideMenu
│   └── ViewController
├── Model/
│   └── User
├── Utilities/
│   └── RoundedCorner
├── Preview Content/
│   └── Preview Assets
└── Assets/
    └── Image assets and resources
```

SwiftUI View Organization

- Organize view content logically (background, main content, buttons)
- Group related elements with clear comment headers
- Use meaningful spacing values
- Include Preview provider at bottom of file

Swift Guidelines

- Use @State for view-local state
- Use @EnvironmentObject for shared data
- Proper view modifier ordering
- Clear view hierarchy structure
- Async/await for asynchronous operations

The focus is on maintaining:
- Clear file organization
- Consistent commenting
- Logical view structure
- Standard SwiftUI conventions
- Clear change tracking