Ollama macOS App - Project Blueprint

Project Structure

```
OllamaApp/
├── OllamaApp.xcodeproj
├── OllamaApp/
│   ├── App/
│   │   ├── OllamaAppApp.swift              # Main app entry point
│   │   └── ContentView.swift               # Root view container
│   │
│   ├── Core/
│   │   ├── Models/                         # Data models
│   │   │   ├── Message.swift
│   │   │   ├── OllamaModel.swift
│   │   │   ├── ChatSession.swift
│   │   │   └── AppSettings.swift
│   │   │
│   │   ├── Services/                       # Business logic services
│   │   │   ├── OllamaAPIService.swift
│   │   │   ├── TextToSpeechService.swift
│   │   │   ├── ModelManagerService.swift
│   │   │   └── SettingsService.swift
│   │   │
│   │   ├── Utilities/                      # Helper utilities
│   │   │   ├── Constants.swift
│   │   │   ├── Extensions/
│   │   │   │   ├── String+Extensions.swift
│   │   │   │   └── View+Extensions.swift
│   │   │   └── NetworkError.swift
│   │   │
│   │   └── DependencyInjection/            # DI container
│   │       └── ServiceContainer.swift
│   │
│   ├── Features/
│   │   ├── Chat/                           # Chat feature module
│   │   │   ├── Views/
│   │   │   │   ├── ChatView.swift
│   │   │   │   ├── MessageBubble.swift
│   │   │   │   ├── MessageInputView.swift
│   │   │   │   └── ChatScrollView.swift
│   │   │   ├── ViewModels/
│   │   │   │   └── ChatViewModel.swift
│   │   │   └── Components/
│   │   │       ├── TypingIndicator.swift
│   │   │       └── LoadingSpinner.swift
│   │   │
│   │   ├── ModelManagement/                # Model management feature
│   │   │   ├── Views/
│   │   │   │   ├── ModelSelectorView.swift
│   │   │   │   ├── ModelDiscoveryView.swift
│   │   │   │   ├── ModelDownloadView.swift
│   │   │   │   └── ModelDetailsView.swift
│   │   │   ├── ViewModels/
│   │   │   │   ├── ModelSelectorViewModel.swift
│   │   │   │   └── ModelDiscoveryViewModel.swift
│   │   │   └── Components/
│   │   │       ├── ModelCard.swift
│   │   │       └── DownloadProgressView.swift
│   │   │
│   │   ├── Avatar/                         # Interactive avatar feature
│   │   │   ├── Views/
│   │   │   │   ├── AvatarView.swift
│   │   │   │   └── AvatarControlsView.swift
│   │   │   ├── ViewModels/
│   │   │   │   └── AvatarViewModel.swift
│   │   │   ├── SpriteKit/
│   │   │   │   ├── AvatarScene.swift
│   │   │   │   ├── AvatarNode.swift
│   │   │   │   └── AnimationManager.swift
│   │   │   └── Assets/
│   │   │       ├── Expressions/
│   │   │       │   ├── neutral.png
│   │   │       │   ├── happy.png
│   │   │       │   ├── thinking.png
│   │   │       │   └── speaking.png
│   │   │       └── Animations/
│   │   │           ├── idle.sks
│   │   │           ├── blink.sks
│   │   │           └── talk.sks
│   │   │
│   │   └── Settings/                       # App settings
│   │       ├── Views/
│   │       │   ├── SettingsView.swift
│   │       │   ├── VoiceSettingsView.swift
│   │       │   └── AvatarSettingsView.swift
│   │       └── ViewModels/
│   │           └── SettingsViewModel.swift
│   │
│   ├── Shared/                             # Shared UI components
│   │   ├── Components/
│   │   │   ├── CustomButtons/
│   │   │   │   ├── PrimaryButton.swift
│   │   │   │   └── SecondaryButton.swift
│   │   │   ├── CustomTextFields/
│   │   │   │   └── ChatTextField.swift
│   │   │   └── Modifiers/
│   │   │       ├── ChatBubbleModifier.swift
│   │   │       └── ShadowModifier.swift
│   │   │
│   │   └── Theme/
│   │       ├── AppTheme.swift
│   │       ├── Colors.swift
│   │       └── Typography.swift
│   │
│   └── Resources/
│       ├── Assets.xcassets/
│       │   ├── AppIcon.appiconset/
│       │   ├── Colors/
│       │   └── Images/
│       ├── Localizable.strings
│       └── Info.plist
│
├── OllamaAppTests/
│   ├── CoreTests/
│   │   ├── ServicesTests/
│   │   │   ├── OllamaAPIServiceTests.swift
│   │   │   └── TextToSpeechServiceTests.swift
│   │   └── ModelsTests/
│   │       └── MessageTests.swift
│   │
│   ├── FeaturesTests/
│   │   ├── ChatTests/
│   │   │   └── ChatViewModelTests.swift
│   │   └── ModelManagementTests/
│   │       └── ModelSelectorViewModelTests.swift
│   │
│   └── TestUtilities/
│       ├── MockServices/
│       │   ├── MockOllamaAPIService.swift
│       │   └── MockTextToSpeechService.swift
│       └── TestData/
│           └── SampleData.swift
│
└── OllamaAppUITests/
    ├── ChatUITests.swift
    ├── ModelManagementUITests.swift
    └── AvatarUITests.swift
```

Core Components Blueprint

1. App Entry Point

`OllamaAppApp.swift`

```swift
// Main app entry point
// - Configure dependency injection container
// - Set up app-wide environment objects
// - Handle app lifecycle events
```

`ContentView.swift`

```swift
// Root container view with navigation
// - TabView or NavigationSplitView for main sections
// - Chat, Model Management, Avatar, Settings tabs
// - Handle deep linking and state restoration
```

2. Data Models

`Message.swift`

```swift
// Chat message model
struct Message: Identifiable, Codable {
    let id: UUID
    let text: String
    let sender: MessageSender // .user, .assistant
    let timestamp: Date
    var isStreaming: Bool
    var metadata: MessageMetadata?
}

enum MessageSender: String, CaseIterable {
    case user, assistant
}

struct MessageMetadata {
    let model: String?
    let tokens: Int?
    let processingTime: TimeInterval?
}
```

`OllamaModel.swift`

swift

```swift
// Ollama model representation
struct OllamaModel: Identifiable, Codable {
    let id: String
    let name: String
    let size: Int64
    let modifiedAt: Date
    let digest: String
    var isDownloading: Bool
    var downloadProgress: Double
}

struct AvailableModel: Identifiable, Codable {
    let id: String
    let name: String
    let description: String
    let tags: [String]
    let pullCommand: String
    let size: String
}
```

`ChatSession.swift`

swift

```swift
// Chat session management
struct ChatSession: Identifiable, Codable {
    let id: UUID
    var title: String
    var messages: [Message]
    let createdAt: Date
    var modifiedAt: Date
    let modelName: String
}
```

`AppSettings.swift`

swift

```swift
// App configuration and user preferences
struct AppSettings: Codable {
    var selectedModel: String
    var ollamaBaseURL: String
    var voiceSettings: VoiceSettings
    var avatarSettings: AvatarSettings
    var chatSettings: ChatSettings
}

struct VoiceSettings: Codable {
    var isEnabled: Bool
    var voice: String
    var rate: Float
    var pitch: Float
    var volume: Float
}

struct AvatarSettings: Codable {
    var isEnabled: Bool
    var selectedAvatar: String
    var expressionSensitivity: Float
    var animationSpeed: Float
}

struct ChatSettings: Codable {
    var streamingEnabled: Bool
    var autoSave: Bool
    var maxTokens: Int?
    var temperature: Float?
}
```

3. Services Layer

`OllamaAPIService.swift`

swift

```swift
// Ollama API communication service
class OllamaAPIService: ObservableObject {
    // Properties
    private let baseURL: String
    private let session: URLSession

    // Methods
    func sendMessage(text: String, model: String, history: [Message]) async throws -> 
    func generateCompletion(prompt: String, model: String) async throws -> String
    func getAvailableModels() async throws -> [OllamaModel]
    func pullModel(name: String) async throws -> AsyncThrowingStream<ModelDownloadProg
    func deleteModel(name: String) async throws
    func getModelInfo(name: String) async throws -> ModelInfo

    // Private helpers
    private func handleStreamingResponse(_ data: Data) throws -> String?
    private func createChatRequest(text: String, model: String, history: [Message]) ->
}

struct ModelDownloadProgress {
    let status: String
    let digest: String?
    let total: Int64?
    let completed: Int64?
}
```

TextToSpeechService.swift

swift

```swift
// Text-to-speech functionality using AVFoundation
class TextToSpeechService: NSObject, ObservableObject {
    // Properties
    private let synthesizer: AVSpeechSynthesizer
    @Published var isSpeaking: Bool = false
    @Published var currentWord: String = ""

    // Speech control
    func speak(text: String, voice: String, rate: Float, pitch: Float)
    func stopSpeaking()
    func pauseSpeaking()
    func continueSpeaking()

    // Voice management
    func getAvailableVoices() -> [AVSpeechSynthesisVoice]
    func setVoice(_ voice: AVSpeechSynthesisVoice)

    // Delegate methods for word-level timing
    // Used for avatar lip sync coordination
}

// AVSpeechSynthesizerDelegate extension for timing callbacks
```

`ModelManagerService.swift`

swift

```swift
// Model management and caching
class ModelManagerService: ObservableObject {
    @Published var installedModels: [OllamaModel] = []
    @Published var availableModels: [AvailableModel] = []
    @Published var currentModel: OllamaModel?

    // Model operations
    func refreshInstalledModels() async
    func loadAvailableModels() async
    func downloadModel(_ model: AvailableModel) async throws
    func deleteModel(_ model: OllamaModel) async throws
    func selectModel(_ model: OllamaModel)

    // Progress tracking
    func trackDownloadProgress(for modelName: String) -> AsyncThrowingStream<Double, E
}
```

`SettingsService.swift`

```swift
// App settings persistence and management
class SettingsService: ObservableObject {
    @Published var settings: AppSettings

    // Settings management
    func loadSettings()
    func saveSettings()
    func resetToDefaults()

    // Specific setting updates
    func updateVoiceSettings(_ voiceSettings: VoiceSettings)
    func updateAvatarSettings(_ avatarSettings: AvatarSettings)
    func updateChatSettings(_ chatSettings: ChatSettings)
}
```

4. Feature ViewModels

`ChatViewModel.swift`

```swift
// Main chat functionality coordinator
class ChatViewModel: ObservableObject {
    // Dependencies
    private let apiService: OllamaAPIService
    private let ttsService: TextToSpeechService
    private let settingsService: SettingsService

    // Published properties
    @Published var messages: [Message] = []
    @Published var currentInput: String = ""
    @Published var isLoading: Bool = false
    @Published var errorMessage: String?
    @Published var currentModel: String = ""

    // Chat operations
    func sendMessage()
    func clearChat()
    func regenerateResponse()
    func stopGeneration()

    // Message handling
    private func handleStreamingResponse(_ stream: AsyncThrowingStream<String, Error>)
    private func addUserMessage(_ text: String)
    private func addAssistantMessage(_ text: String)
    private func updateLastAssistantMessage(_ text: String)

    // Integration with TTS and Avatar
    private func speakResponse(_ text: String)
    private func notifyAvatarOfSpeech(_ text: String)
}
```

`AvatarViewModel.swift`

```swift
// Avatar animation and TTS coordination
class AvatarViewModel: ObservableObject {
    // Dependencies
    private let ttsService: TextToSpeechService
    private let settingsService: SettingsService

    // Published properties
    @Published var currentExpression: AvatarExpression = .neutral
    @Published var isSpeaking: Bool = false
    @Published var isEnabled: Bool = true

    // Avatar control
    func setExpression(_ expression: AvatarExpression)
    func startSpeaking(text: String)
    func stopSpeaking()
    func playIdleAnimation()

    // Expression analysis
    private func analyzeTextForExpression(_ text: String) -> AvatarExpression
    private func scheduleExpressionChange(_ expression: AvatarExpression, delay: TimeI

    // SpriteKit scene communication
    private func updateAvatarScene()
}

enum AvatarExpression: String, CaseIterable {
    case neutral, happy, thinking, speaking, surprised, confused
}
```

5. SpriteKit Avatar System

`AvatarScene.swift`

```swift
// Main SpriteKit scene for avatar rendering
class AvatarScene: SKScene {
    // Nodes
    private var avatarNode: AvatarNode?
    private var backgroundNode: SKSpriteNode?

    // Animation state
    private var currentExpression: AvatarExpression = .neutral
    private var isSpeaking: Bool = false

    // Scene setup
    override func didMove(to view: SKView)

    // Public interface
    func setExpression(_ expression: AvatarExpression, animated: Bool = true)
    func startSpeakingAnimation()
    func stopSpeakingAnimation()
    func playIdleAnimation()
    func setMouthOpenness(_ openness: Float) // For lip sync

    // Animation helpers
    private func createExpressionAction(for expression: AvatarExpression) -> SKAction
    private func createSpeakingAction() -> SKAction
    private func createIdleAction() -> SKAction
}
```

AvatarNode.swift

```swift
// Individual avatar sprite node with animations
class AvatarNode: SKSpriteNode {
    // Animation components
    private var faceNode: SKSpriteNode
    private var eyesNode: SKSpriteNode
    private var mouthNode: SKSpriteNode
    private var eyebrowsNode: SKSpriteNode

    // Animation state
    private var currentMouthShape: MouthShape = .closed
    private var blinkTimer: Timer?

    // Initialization
    init(avatarType: AvatarType)

    // Animation methods
    func animateToExpression(_ expression: AvatarExpression, duration: TimeInterval)
    func setMouthShape(_ shape: MouthShape)
    func blink()
    func startIdleAnimations()
    func stopIdleAnimations()

    // Asset management
    private func loadAssets()
    private func createAnimationActions()
}

enum MouthShape: String, CaseIterable {
    case closed, open, smile, speak1, speak2, speak3
}

enum AvatarType: String, CaseIterable {
    case robot, human, cartoon
}
```

6. Views Architecture

`ChatView.swift`

swift

```swift
// Main chat interface
struct ChatView: View {
    @StateObject private var viewModel: ChatViewModel
    @EnvironmentObject private var settingsService: SettingsService

    var body: some View {
        VStack {
            // Chat messages area
            ChatScrollView(messages: viewModel.messages)

            // Input area
            MessageInputView(
                text: $viewModel.currentInput,
                onSend: viewModel.sendMessage,
                isLoading: viewModel.isLoading
            )
        }
        .navigationTitle("Chat")
        .toolbar { /* toolbar items */ }
        .alert("Error", isPresented: .constant(viewModel.errorMessage != nil)) {
            // Error handling
        }
    }
}
```

AvatarView.swift

```swift
// Avatar display and controls
struct AvatarView: View {
    @StateObject private var viewModel: AvatarViewModel
    @State private var avatarScene: AvatarScene

    var body: some View {
        VStack {
            // SpriteKit view
            SpriteView(scene: avatarScene)
                .frame(height: 300)
                .cornerRadius(12)

            // Avatar controls
            AvatarControlsView(viewModel: viewModel)
        }
        .onAppear { setupAvatar() }
        .onChange(of: viewModel.currentExpression) { newExpression in
            avatarScene.setExpression(newExpression)
        }
        .onChange(of: viewModel.isSpeaking) { isSpeaking in
            if isSpeaking {
                avatarScene.startSpeakingAnimation()
            } else {
                avatarScene.stopSpeakingAnimation()
            }
        }
    }

    private func setupAvatar() {
        // Initialize avatar scene
    }
}
```

7. Dependency Injection

`ServiceContainer.swift`

```swift
// Central dependency injection container
class ServiceContainer: ObservableObject {
    // Singleton services
    lazy var ollamaAPIService: OllamaAPIService = {
        OllamaAPIService(baseURL: settingsService.settings.ollamaBaseURL)
    }()

    lazy var textToSpeechService: TextToSpeechService = {
        TextToSpeechService()
    }()

    lazy var modelManagerService: ModelManagerService = {
        ModelManagerService(apiService: ollamaAPIService)
    }()

    lazy var settingsService: SettingsService = {
        SettingsService()
    }()

    // Factory methods for ViewModels
    func makeChatViewModel() -> ChatViewModel {
        ChatViewModel(
            apiService: ollamaAPIService,
            ttsService: textToSpeechService,
            settingsService: settingsService
        )
    }

    func makeAvatarViewModel() -> AvatarViewModel {
        AvatarViewModel(
            ttsService: textToSpeechService,
            settingsService: settingsService
        )
    }

    func makeModelSelectorViewModel() -> ModelSelectorViewModel {
        ModelSelectorViewModel(
            modelManagerService: modelManagerService,
            settingsService: settingsService
        )
    }
}
```

Development Phases

Phase 1: Core Chat (Weeks 1-2)

- Implement basic models and API service
- Create chat view and view model
- Add streaming response handling
- Basic error handling and loading states

Phase 2: Model Management (Week 3)

- Implement model manager service
- Create model selector and discovery views
- Add download progress tracking
- Model switching functionality

Phase 3: Avatar & TTS (Weeks 4-6)

- Implement text-to-speech service
- Create SpriteKit avatar system
- Add basic lip sync with speech timing
- Integrate expression changes based on content

Phase 4: Polish & Advanced Features (Ongoing)

- Improve avatar animations and expressions
- Add more sophisticated lip sync
- Settings and customization options
- Performance optimization and testing

Key Architectural Decisions

1. **MVVM Pattern**: Clear separation of concerns, testable ViewModels
2. **Dependency Injection**: Centralized service management, easy testing
3. **Combine Framework**: Reactive programming for data streams
4. **SwiftUI + SpriteKit**: Native performance with advanced graphics
5. **Async/Await**: Modern concurrency for API calls
6. **Modular Structure**: Feature-based organization for scalability

This blueprint provides a solid foundation for building a sophisticated, native macOS Ollama client with advanced interactive features while maintaining clean architecture and testability.