

Klariti OS iOS App Prototype – Developer Brief

Overview

Klariti OS is an **iOS app prototype** designed to help users reclaim their time and build healthier digital habits. It does this by combining **physical NFC tags** and a **social accountability partner system** to encourage focused, distraction-free phone use. The prototype's core functionality centers on:

- **Physical focus activation:** Using NFC tags as a “*focus mode*” switch – tapping a tag can start (or eventually stop) a focus session.
- **Social accountability:** Letting users add friends as accountability partners and issue **focus challenges** to each other.
- **Distraction blocking (simulated):** During a focus session, the app imitates blocking of other distracting apps to keep the user on task.

Functional Requirements

- **User Registration & NFC Tag Setup:**

- Users can create an account or sign in (basic authentication) to identify themselves in the app.
- After registration, the user is prompted to set up an NFC tag. The app will use **Core NFC** to scan and write to a physical NFC tag:
 - The tag is linked to the user’s profile (e.g., writing a unique identifier or a specific payload to the tag).
 - This tag will serve as the physical trigger for focus mode. (For example, tapping the phone on the tag can initiate a focus session.)
- The app should handle cases where an NFC tag is not immediately available (allowing setup later in settings).

- **Friend/Accountability Partner System:**

- Users can **add friends** within the app to serve as accountability partners. This may involve searching for a username/email or accepting friend requests.
- Once two users are friends, they can see each other’s online status or recent focus activity (if implemented) and can send focus challenges to each other.
- The system should include basic friend management: sending requests, accepting/declining invitations, and listing confirmed friends. (For the prototype, a simple implementation is fine — e.g. no extensive social features beyond what’s needed for challenges.)
- **Accountability aspect:** Friends will be notified of each other’s focus sessions/challenges, adding social pressure to stay on task.

- **Focus Challenge Mechanics:**

- **Creating a Challenge:** A user can initiate a focus challenge targeting a friend. This involves selecting a friend from their list and defining the challenge parameters (e.g. a focus duration like 30 minutes, or a start time).

- **Sending & Notification:** When user A sends a challenge to user B, user B should receive a notification or in-app alert with the details (who challenged them and for how long).
 - **Accept/Decline:** User B can accept the challenge (triggering focus mode for both participants) or decline it. If declined, user A should be notified and the challenge is canceled.
 - **Focus Session Start:** If the challenge is accepted, both users' apps enter **Focus Mode** simultaneously:
 - A countdown timer or other UI indicator shows the remaining focus time.
 - Both users are expected to refrain from using other apps during this period (reinforced by the app's simulated blocking – see below).
 - **Completion Tracking:** When the timer ends, both users are notified of a successful completion. The app can log the session (e.g., for stats or future features like "weekly focus streaks").
 - If a user quits the focus mode early (breaks the challenge), the app should mark the challenge as failed for that user and ideally notify the partner. (Since actual app usage can't be enforced, this relies on the honor system or detecting that the app went to background; see limitations.)
 - **Solo Focus (optional):** Even without a friend, a user should be able to start a focus session on their own (perhaps by tapping their NFC tag or via an app button) for personal use. This would use the same focus mode mechanics but without the challenge notification flow.
- **Simulated Distracting App Block (Focus Mode):**
 - When Focus Mode is active (whether from a challenge or a solo session), the app will **simulate the blocking of other apps** to help the user stay focused. Because iOS doesn't allow third-party apps to actually lock the device or block other apps, we achieve this through design and limited system features:
 - The app will present a **full-screen focus interface** (e.g., a timer, motivational message, and perhaps an "End Focus" button) to discourage leaving the app.
 - Ideally, the prototype can **detect if the user navigates away** (app moves to background). While it can't prevent it, it might use app lifecycle events to record if a user left early. For instance, if the app is backgrounded before the timer ends, we flag that the focus was interrupted.
 - We might use a local notification or alert to immediately remind the user to return to focus if they switch out (as a gentle nudge, since actual blocking isn't possible).
 - The concept of "blocking distracting apps" can be demonstrated by *not* allowing navigation within the focus app to any other screen without ending the session, and by instructing the users (and showing in the UI) that social media or other specified apps are "blocked" during this time. (In a real implementation, one might integrate iOS Screen Time APIs or Shortcuts to disable apps, but for this prototype it's only a visual/behavioral simulation.)
 - Ending focus mode: The user (or both users in a challenge) can end the session when the timer is up, or manually via an **End Focus** action (with a confirmation prompt to prevent accidental exits). Ending early would count as a break/failure in the context of a challenge.
 - **Summary:** The focus here is to mimic the experience of a locked-down phone without implementing actual OS-level restrictions. This keeps the user experience clear for demonstration purposes while staying within iOS sandbox limitations.

Technical Considerations

- **Platform & Devices:**

- Target **iOS** (iPhone only) – the prototype will run on modern iPhones that support NFC (iPhone 7 and later, running iOS 14+).
- Ensure compatibility with the latest iOS version (test on iOS 17, for example). iPads are out of scope since most iPads lack NFC, and the use-case is phone-centric.
- The app will be built with Swift (and UIKit or SwiftUI as appropriate – internal decision) and utilize Apple's Core frameworks (e.g., Core NFC).

- **Core NFC Integration:**

- The app will use **Core NFC** to read/write NFC tags. Remember to include the appropriate entitlements and privacy usage descriptions (**NFCReaderUsageDescription**) in the app's Info.plist.
- **Reading/Writing Tags:** Use NFC NDEF messages to store data on the tag:
 - When linking a tag to a user, we can write a small NDEF record containing a unique identifier or a URL scheme that our app can recognize. For example, an NDEF text record with something like **klariti:[userID]** or a custom URI.
 - For simplicity, the prototype might just read the tag's UID (unique hardware ID) and associate that with the user's profile in the app database, instead of writing data. Writing is optional if reading the UID suffices to identify the tag.
- **Triggering Focus via Tag:** When the user scans the linked NFC tag:
 - If the app is in the foreground, we initiate the focus mode immediately upon a successful scan.
 - *Note:* iOS allows certain NFC tags to launch apps in the background via App Clips or Universal Links in the tag's payload. Implementing background launch is advanced and may require specific NFC tag setup (e.g., an NDEF with a deep link). For the prototype, it's acceptable that the app must be open (or a scan initiated from within the app) to start focus mode.
- The NFC operations should be robust: handle errors like tag reading failure or if the wrong tag is scanned (e.g., show an error if the tag isn't recognized or prompt to set it up if not done).

- **Networking & Data:**

- To support the friend system and challenges, a basic backend or peer-to-peer communication will be needed. For the prototype, options include:
 - Using a lightweight cloud database or real-time service (e.g., Firebase, CloudKit) to store user accounts, friend relationships, and pending challenges.
 - Alternatively, simulating this by running two instances of the app (on two devices) with test accounts and using push notifications or a simple server to relay challenge invites.
- **Notifications:** If possible, use **Push Notifications** to inform a user of a challenge invite or a friend request. This requires setting up APNs for the app. If that's too complex for the prototype timeline, consider in-app notifications (if both users are online) or a simple polling mechanism to fetch new challenges when the app is opened.
- **Data Storage:** User data (like friend list, tag ID, and focus logs) can be stored locally (Core Data or simple in-memory for the prototype) and/or on the backend. Security is not a primary concern for the prototype, but basic measures (like not storing plain text passwords if accounts are implemented) should be followed.

- **iOS Focus/Screen Time APIs:**

- Be aware of existing iOS features like Focus modes and Screen Time restrictions. The prototype will **not directly integrate** with these due to complexity and permission constraints, but developers should know why:
 - Changing system focus or app restrictions programmatically is not allowed for third-party apps (except maybe through MDM or enterprise profiles, which we won't use).
 - Thus, our app's focus mode remains an in-app experience only.
- This means the development should concentrate on the app's UI/UX to keep users engaged during focus time, rather than trying to hack around OS limitations.

- **Device Handling & Edge Cases:**

- Ensure the app handles interruptions gracefully. For example, if a phone call comes in during a focus session, iOS will foreground the Phone app. Our app should on return detect that focus was paused/interrupted (perhaps pausing the timer) and either allow resumption or mark the session as broken if that's the intended behavior.
- If the user locks the screen during focus mode (which is actually fine and perhaps expected during a focus session), the session should continue running its timer in the background if possible. (Consider using a short **BackgroundTask** to keep the timer, or at least save state and use local notifications to mark end of session if the app isn't foreground at that time.)
- NFC scanning requires the app in foreground and the device unlocked. We should handle the case where the user tries to scan without proper state (e.g., app not active) with appropriate messaging.

Prototype Scope and Limitations

- **Scope Focus – Core Flows Only:** This prototype is meant to demonstrate the *concept* of Klariti OS rather than be a full product. Development will concentrate on the **happy path** for the two main flows (NFC-triggered focus, and friend challenge focus). Non-essential features or polish (fancy UI, extensive settings, etc.) are limited or omitted. The goal is to have a working demo of the idea.
- **Simulated Behaviors:** Many system-level controls are **simulated** due to platform limitations:
 - **App Blocking:** As noted, the prototype cannot actually block other apps. We simulate it by keeping the user within our app and using psychological deterrents. There will be no modifications to iOS Screen Time or actual app lockouts.
 - **Notifications/Backend:** If a full push notification system or backend server is too complex for the prototype timeframe, we will simulate friend notifications (e.g., having both test devices logged in and pre-coordinating challenge start). In an internal demo, developers might trigger the friend's app manually to mimic a notification received. This is acceptable for now.
 - **Single-Device Demo Mode:** In absence of two devices, we might include a "demo mode" toggle to simulate the friend accepting a challenge on the same device for presentation purposes. (For example, the app could have a testing function that instantly accepts your own challenge to move into focus mode, to show the flow without a second phone.)
- **Limitations:**

- The prototype will not cover user account recovery, complex friend search functionality, or any social features beyond friend requests and challenges.
 - Only one NFC tag per user is expected in this version. We won't support multiple tags or complex tag management (though the code could be designed to allow extension later).
 - The UI will be basic and functional. Design consistency and aesthetics are secondary; however, it should clearly convey when the user is in focus mode vs normal mode.
 - **No Persistence Guarantees:** Data might not persist between app launches beyond what's easy to implement (e.g., in-memory or simple local storage). If the app is reinstalled or if there's no real backend, friend lists and accounts may reset – acceptable for a demo.
 - **Testing Constraints:** Since NFC is hardware-dependent, testing in Simulator is not possible for that part. Devs will need to test on a physical iPhone. Plan for internal testing accordingly (have NFC tags on hand, use TestFlight or dev builds on devices).
- **Security/Privacy:** Not a primary focus for the prototype. We'll follow basic best practices (like requiring user permission for NFC and handling any personal data respectfully), but features like encryption, secure login, or privacy settings are not in scope at this stage.

Example User Flows

Below are two key user journeys that the prototype will support, step-by-step:

1. Focus Challenge Flow (User A <-> User B)

1. **Challenge Initiation (User A):** User A opens the app, navigates to their friends list, and chooses a friend (User B) to challenge. They select a focus duration (e.g., 30 minutes) and send a focus challenge invitation.
2. **Challenge Notification (User B):** User B receives an alert from the app: "User A has challenged you to a 30-minute Focus session." They open the app to view the challenge details.
3. **Accepting the Challenge:** User B taps "Accept". (If they tap "Decline", the flow ends here with User A being notified of the decline.)
4. **Focus Mode Begins:** Upon acceptance, both User A and User B's apps automatically enter Focus Mode:
 - A countdown timer (30:00) is displayed on both devices, and perhaps a message like "Focus session in progress...".
 - Normal app navigation is disabled or hidden; the only option might be an "End Focus" button (with a warning).
 - Both users know they're now simultaneously in focus mode, fostering mutual accountability.
5. **During Focus Session:** Both users keep the app open and concentrate on offline tasks. The app might occasionally update the timer on screen. If either user attempts to leave the app, the app will note the interruption (and could remind them to return). They also know their friend is counting on them, adding incentive to stay off other apps.
6. **Completion or Early Exit:**
 - If the full duration passes without interruption, a success message is shown: "Focus complete! 🎉 You and User A stayed focused for 30 minutes." Both users might see each other's success status.
 - If one user ends the session early (by hitting "End Focus" or by leaving the app for too long), the app could mark the challenge as failed for that user: "Focus session ended early." Optionally,

User B might get a notification like “User A left the focus session early” (and vice versa) to enhance accountability.

7. **Post-Session:** The app returns to normal mode. Possibly, a summary screen could show stats (duration achieved, etc.). The friendship aspect could include sending a thumbs-up or encouragement message to each other after completing the challenge.

(Throughout this flow, consider the communication layer: sending the invite, the acceptance, and any early-exit notification require network messaging between A and B's devices. In the prototype, this is simplified but should be represented in the UI logic.)

2. NFC-Triggered Focus Mode Flow (Solo Focus Session)

1. **Initial Setup:** (One-time) User has already registered and linked an NFC tag to their app. For example, they placed an NFC sticker on their desk and paired it during setup. They also possibly configured a default focus duration for tag-initiated sessions (say 60 minutes by default).
2. **Starting Focus via NFC:** When the user is ready to focus, instead of navigating menus in the app, they simply tap their iPhone on the NFC tag (either with the app open, or by launching a “Start Focus” scan within the app).
 - The app detects the tag (via Core NFC). It recognizes the tag’s ID or content as the one linked to this user.
 - Immediately, the app prompts, “Start a 60-minute focus session now?” (If a confirmation is desired; or it could start instantly without extra taps, based on design).
3. **Focus Mode Active:** Upon confirmation, the app enters the Focus Mode screen:
 - The 60-minute timer begins counting down.
 - The interface might display “Focus Mode – Started by NFC tag” to reinforce the physical trigger concept. All distraction-blocking measures (simulated) are in effect just as in the challenge scenario (restricted UI, etc.).
 - Since this is a solo session, no friend is notified or involved by default. However, the user could have the option to “invite a friend to join” even after starting, if that makes sense for extension (not core to prototype though).
4. **During the Session:** The user stays off other apps and keeps the focus. The phone can be put aside or face-down since they know it’s in focus mode. (If the user does switch out, the same logic of recording an interruption applies.)
5. **Ending the Session:** When the timer ends, the app signals completion (“Focus session complete!”). If the user needs to stop early, they use the End Focus button with confirmation.
6. **Reset for Next Use:** The focus session ends and the app returns to the main screen. The user can repeat the process whenever they want to focus again: just tap the NFC tag to re-enter focus mode. The NFC tag essentially serves as a **physical commitment device** — e.g., the user might place it on a spot designated for focused work, and tapping it becomes a ritual to start focusing.