

Timer Focus Mode and Hold-Cancel Work Report

#Overview

Today the standalone timer screen was reworked further with a focus on the active sprint layout, countdown ownership, and the hold-to-cancel interaction.

The main direction was to make the running timer feel more like a focused study state instead of a duration picker that happens to count down. The countdown was moved toward a separate overlay, the task details were given more visual emphasis, and the cancel interaction was changed from a simple button press into a deliberate hold action.

#ImplementedFeatures

#CountdownOverlay

Moved the active countdown away from the duration picker:

- Removed the old selected-picker countdown state
- Added a separate countdown overlay using 'countdownAnimation'
- Added 'focusModeAnimation' so the countdown can move from the central timer area toward the upper-left area
- Kept the picker responsible for duration values only

This separates two responsibilities that had previously been mixed together: the picker selects a duration, while the overlay shows active countdown time.

#FocusModeLayout

Adjusted the active timer layout to put more attention on the task:

- Moved task details higher and closer to the center of the running screen
- Increased the task title and description size
- Kept task details animated through 'taskDetailsAnimation'
- Continued using the red screen overlay as the main visual timer-progress element

The intent is for the active state to feel more like a study-session spotlight, where the selected task becomes the main focus and the countdown becomes supporting information.

#HoldToCancel

Changed the cancel action into a hold interaction:

- Added 'HOLD_TO_CANCEL_MS'
- Added 'cancelHoldTimeoutRef'
- Added a hold-completion haptic warning
- Kept the cancel button scale feedback during press
- Changed the label to 'Hold to end sprint'

This makes cancellation more deliberate and reduces the chance of accidentally ending a sprint with a single tap.

#CancelAccelerationExperiment

Implemented the red timer overlay as cancel feedback:

- Added delayed cancel acceleration through 'CANCEL_ANIMATION_DELAY_MS'
- Added 'cancelHoldAnimationDelayRef'
- Added 'cancelAccelStartedRef' to distinguish quick taps from actual hold acceleration
- Split normal timer progress into 'progressAnimationRef'
- Added 'startProgressAnimation(fromY)' so progress can start or resume from a specific overlay position
- Added 'cancelOverlayAnimation' as a temporary visual offset on top of the real timer progress
- Added 'getCancelOverlayTarget(...)' to calculate how far the cancel preview should move
- Added a release handoff animation so the cancel offset eases back into the real timer position
- Added clamping so the visual overlay does not move past the finished timer position
- Added easing constants for the cancel delay, release handoff, and timer reset timings

The goal was for the red overlay to speed toward the finished position during a hold, then return smoothly to the real timer progress if the user releases before the cancel completes. The

important change is that cancel preview motion is now layered on top of the real progress instead of directly taking over the main timer animation.

#DurationPickerCleanup

Cleaned up the duration picker after moving countdown ownership out of it:

- Removed selected countdown rendering from the picker item
- Kept picker items rendering plain timer values
- Kept picker values fading out during active timer mode
- Added index clamping when reading the selected duration from `'onMomentumScrollEnd'`
- Restored `'duration'` as a dependency of the start callback so the selected picker value is used correctly

This fixed the earlier issue where the timer could behave as if the selected duration was still the initial value.

#TimerCodeCleanup

Cleaned up the timer screen structure after the interaction behavior was stabilized:

- Renamed the old `'animation'` callback to `'startTimer'`
- Renamed unclear animated values like `'opacity'` and `'translateY'` to `'startButtonOpacity'` and `'startButtonTranslateY'`
- Grouped refs by purpose: animated values, timer/session refs, and cancel-hold refs
- Extracted `'clearCountdown'`, `'clearCancelHoldTimers'`, and `'stopTimerAnimations'`
- Extracted the cancel overlay target calculation into `'getCancelOverlayTarget(...)'`
- Split the render section into local render helpers for the overlay, start button, cancel button, countdown, duration picker, and task details
- Moved the timer item layout into `'styles.timerItem'`

This did not change the screen into a separate hook or split the timer into multiple files. The cleanup stayed local to `'timer.tsx'` so the current animation work remains easy to inspect.

#LearningNotes

#AnimationOwnership

The main lesson today was that an `'Animated.Value'` should have one clear owner at a time.

The red overlay now combines two animated values:

- normal timer progress
- hold-to-cancel visual offset

The normal timer progress is controlled by `'timerAnimation'`, while cancel preview motion is controlled by `'cancelOverlayAnimation'`. This avoids stopping the real timer progress just to show the cancel speed-up effect.

#RefsAsMutableState

Several refs were added to track animation and timer ownership:

- `'progressAnimationRef'` tracks the long-running red overlay progress animation
- `'sessionStartedAtRef'` tracks the progress timeline used for recovery calculations
- `'sessionDurationMsRef'` stores the current timer duration in milliseconds
- `'cancelHoldTimeoutRef'` tracks when hold cancellation should complete
- `'cancelHoldAnimationDelayRef'` tracks when cancel acceleration should begin
- `'cancelAccelStartedRef'` tracks whether the red overlay acceleration actually started
- `'cancelHoldActiveRef'` and `'cancelHoldIdRef'` prevent stale delayed hold callbacks from taking over after release

The important distinction is that assigning to `'current'` is allowed even when the ref variable itself is declared with `'const'`.

#CancelOffsetHandoff

The release recovery logic was changed to avoid rewriting the real timer progress:

- keep `'timerAnimation'` running as the source of real timer progress
- add `'cancelOverlayAnimation'` on top of it while the cancel button is held
- animate only the cancel offset back to `'0'` when the hold is released

- keep the visible overlay clamped to the screen height
- tune the release handoff timing with 'CANCEL_RELEASE_MS'

This makes the visual red overlay return to the countdown's real timer position without forcing the main timer animation to stop and restart.

#CurrentState

The hold-cancel red overlay interaction has been reworked so the cancel preview no longer directly mutates the real timer progress.

The current implementation:

- keeps the countdown and real timer progress owned by 'timerAnimation'
- uses 'cancelOverlayAnimation' as a temporary visual offset during hold-to-cancel
- invalidates stale hold callbacks with 'cancelHoldIdRef'
- eases the cancel offset back to '0' on release
- keeps the cancel-completion path separate from normal timer completion

This should make the red overlay speed-up feel connected to the cancel hold while still keeping the timer progress visually aligned with the countdown after release.

#Verification

Current static checks pass:

```
""text
npm run lint
exited successfully
""
```

```
""text
npx tsc --noEmit
exited successfully
""
```

The hold-cancel handoff was also adjusted based on runtime feedback so the cancel offset eases back more smoothly into the real timer progress.

#FilesChanged

Main file worked on:

```
""text
app/(tabs)/timer.tsx
""
```

New note added:

```
""text
notes/work-report-timer-2026-04-24.md
""
```

#Conclusion

The timer screen moved further toward a focused active-sprint experience. The countdown is now separated from the duration picker, task details have more visual weight, and cancel is treated as a deliberate hold action rather than a normal tap.

The main animation change is that hold-to-cancel now keeps the real timer progress separate from the temporary cancel speed-up effect. The code was also cleaned up so the timer flow is easier to read and continue working on.

Problems occurring after writing conclusion

Tried to implement sound by installing expo-audio. This caused the dependency list to update. The

diff was massive, and something in the diff caused the entire timer page to break. Logic, animations - the lot. Have reverted back to last known working dependency list, as well as un-refactored a lot of code in an attempt to revert to a functioning state before figuring out that the culprit was dependencies. Need to figure out what is causing the critical failure in the new list.

Todo

- Re-refactor to make code cleaner, more readable and easier to maintain.
- Figure out the dependency issues of later dependency lists

Conclusion of dependency saga

There was a mismatch in the nativewind dependency, with my one being ^4.2.3 and the other list being ^4.1.23. This caused my entire timer screen to fail. Animations got borked, buttons not working properly, duration picker only showing 2 indexes... the works. Solution - keep nativewind dependency to ^4.2.3