

Timer Interaction and Cancel Flow Work Report

#Overview

Today the standalone timer screen was developed further with a focus on the cancel interaction, countdown reset order, and a progress cue inside the cancel button.

The main work was not just adding UI pieces, but understanding how the existing React Native 'Animated' flow behaves when a timer is started, cancelled, or allowed to finish naturally. The timer is still being treated as its own tab with placeholder task information, but the interaction model is now closer to the intended study-session behavior.

#ImplementedFeatures

#CancelButton

Added a dedicated cancel control for the active timer state:

- Added a separate cancel button animation value
- Added a bottom-positioned cancel button that appears only during the running state
- Added reverse handling so the button can be dismissed again when cancelling manually or when the timer finishes

The main goal was to keep the original large start control as the primary entry point, while giving the active timer state its own secondary exit action.

#CancelProgressCue

Started adding a progress cue directly inside the cancel button:

- Added a separate 'cancelProgressAnimation'
- Added an inner animated fill layer inside the cancel button
- Changed the progress direction to move left-to-right inside the button instead of using a full-button opacity fade

This was done to match the visual language of the main red timer overlay while keeping the progress indicator smaller and more local to the cancel action.

#DurationLocking

Updated the duration selector to stay fixed while the timer is running:

- Added 'scrollEnabled={!timerIsRunning}' to the horizontal timer picker
- Added an early return inside 'onMomentumScrollEnd'
- Prevented the selected timer duration from changing once a session has started

This keeps the timer state consistent after the session begins and avoids the picker drifting into a visually different value while the countdown is active.

#CountdownOwnership

Clarified how the countdown interval should be owned and reset:

- Added 'countdownRef'
- Added interval clearing before starting a new countdown
- Used the ref-based interval handle so cancel and finish logic can target the active countdown

This work was needed because countdown behavior becomes unreliable if the code starts new intervals without keeping a consistent reference to the currently running one.

#CancelFlowSequencing

Worked on the ordering of reverse animations during manual cancel:

- Tested separating countdown fade-out from the picker/start-button return
- Investigated why adjacent numbers were reappearing before the countdown text had fully finished reversing
- Traced the problem to both animation timing and the 'showCountdownText' render condition

The important lesson here was that hiding the countdown visually and switching the rendered text

back to the normal timer value are related, but not identical, events.

#LearningNotes

#AnimatedValueResponsibilities

Today reinforced that each 'Animated.Value' should have one clear responsibility:

- 'timerAnimation' controls the red overlay position
- 'buttonAnimation' controls start-button disappearance and inactive picker return
- 'countdownAnimation' controls countdown visibility
- 'cancelButtonAnimation' controls the cancel button itself
- 'cancelProgressAnimation' controls the left-to-right fill inside the cancel button

Several visual bugs came from trying to make one animated value carry two different meanings at the same time.

#RenderStateVsAnimationState

A key distinction became clearer during the cancel-flow debugging:

- Animated values control motion and opacity
- Regular React state controls what text/content is actually rendered

One important example is 'showCountdownText':

- Even if the countdown has visually faded out, the selected timer item still renders 'MM:SS' while 'showCountdownText' remains 'true'
- This means the UI can still appear to be in 'countdown mode' even after part of the reverse animation has already completed

This is why some cancel-order issues were not purely animation problems.

#SequenceVsParallel

The timer work also clarified when 'Animated.sequence([...])' and 'Animated.parallel([...])' should be used:

- 'sequence' is for strict order
- 'parallel' is for animations that should run at the same time

One mistake that surfaced during the progress-button work was placing the long progress-fill animation in a sequence after the main timer animation, which caused the fill to begin only after the timer had already ended.

#CurrentIssue

The current timer screen still has remaining cancel-flow polish issues around visual timing and overlay cleanup.

The main issue currently under investigation is the reset order during manual cancel:

- the red timer overlay can still produce a visible flash/jump when the running animation is interrupted
- the adjacent picker numbers and selected countdown text are sensitive to both animation order and 'showCountdownText'
- the current implementation needs further refinement so cancel feels deliberate instead of visually noisy

Current lint result:

```
““text
npm run lint
completed with 1 warning
““
```

Current warning:

- unnecessary 'showCountdownText' dependency in one 'useCallback'

There are no current lint errors, but the cancel interaction is not yet considered visually finished.

#FilesChanged

Main file worked on:

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

New note added:

```
““text
notes/work-report-timer-2026-04-23.md
““
```

#Conclusion

The timer screen moved further toward a complete active-session interaction today. It now has a dedicated cancel control, a left-to-right progress cue inside that control, a locked duration picker while running, and a clearer separation between countdown ownership and animation ownership.

The main remaining work is not basic feature addition, but interaction polish. In particular, the cancel sequence still needs refinement so the red overlay, countdown text, and adjacent timer values return in a clean and intentional order.