

Task Timer Integration and App Polish Work Report

#Overview

Today the timer work moved from being a standalone tab experiment into the actual task workflow.

The main commit used for this summary is:

```
““text
c74062c Implemented timer into task details, uploaded example images for app and centred headers
on all screens
““
```

The work focused on connecting the sprint timer to individual tasks, preserving an active sprint locally, cleaning up routing, and polishing the surrounding app presentation. The timer is no longer exposed as a top-level tab. It now belongs to the task details flow where a sprint naturally starts from a selected task.

#ImplementedFeatures

#TimerRouteIntegration

Moved the timer route from the tab navigator into the task stack:

- removed the old timer tab from 'app/(tabs)/_layout.tsx'
- added 'timer' as a screen in 'app/task/_layout.tsx'
- moved the timer implementation from 'app/(tabs)/timer.tsx' to 'app/task/timer.tsx'
- added a 'Start Sprint' action from the task details screen
- passed the selected task id into the timer route using 'tId'

This makes the timer part of the task workflow instead of a separate global screen.

#TaskAwareTimer

Updated the timer screen so it can load and display the selected task:

- reads 'tId' from route params
- fetches the matching task from Supabase
- shows the task title and description during the sprint
- falls back to generic sprint text if task data is missing

This replaces the earlier placeholder-task model and makes the sprint screen reflect the actual task being worked on.

#ActiveSprintPersistence

Added local persistence for the current sprint in 'lib/asyncStorage.ts':

- added an 'ActiveSprint' type
- added 'SaveActiveSprint(...)'
- added 'GetActiveSprint()'
- added 'RemoveActiveSprint()'
- stores the active task id, sprint duration, and calculated end time

The timer now saves the sprint end time when a session starts. When the timer screen is reopened for the same task, it can restore the remaining sprint time instead of treating the session as gone.

#TimeBasedCountdown

Changed the countdown ownership toward wall-clock time:

- calculates 'endTime' when the sprint starts
- updates remaining time from 'Date.now()'
- restores progress from elapsed time when an active sprint is found
- removes the active sprint when it expires or is cancelled

This is a step toward making the timer more robust when the app is backgrounded or the timer screen is reopened.

#HoldCancelOverlayWork

Continued work on the red hold-to-cancel timer overlay:

- kept 'timerAnimation' as the main timer-progress value
- kept 'cancelOverlayAnimation' as the temporary hold-preview offset
- added measured overlay height handling through 'containerHeight'
- added an offscreen reset position for the red overlay
- added 'timerOverlayVisible' so the red overlay can be hidden immediately after manual cancel fires

The final direction was to stop relying only on moving the red overlay offscreen. The cancel path now also hides the overlay by opacity before the rest of the return animations run.

#HeaderAlignmentPolish

Centered navigation titles across the main app screens:

- dashboard
- subjects
- subject create/edit
- subject details
- assignment create/edit
- assignment details
- task create/edit
- task details
- sprint timer

This was a small visual consistency pass, but it makes the app feel less uneven between screens.

#ImageAssetUpdate

Updated the app image assets:

- replaced the main icon and splash image files under 'assets/images/'
- moved 'master.png' into 'assets/images/'
- removed the older 'assets/study-sprint-image-pack/' copies

This keeps the active image assets in the folder Expo expects instead of keeping a separate image-pack folder around.

#ProblemsAndSetbacks

#CancelOverlayBug

The main setback today was the red hold-to-cancel overlay still being visible after manual cancel.

Several possible causes were investigated:

- the overlay using 'Dimensions.get('window')' instead of the measured container height
- the overlay being clamped to the screen height
- the overlay view being stretched by 'StyleSheet.absoluteFillObject'
- the overlay not being moved far enough below the screen

The first fixes improved the logic but did not remove the visible red bar in runtime feedback. The latest approach adds explicit overlay visibility state so the red timer layer can be hidden directly when cancel completes.

#ManualCancelVsNaturalFinish

Another important clarification was that the problematic path was manual cancel, not the natural "timer reached zero" flow.

The cancel path has extra moving parts:

- hold delay
- hold completion timeout
- 'cancelOverlayAnimation'
- haptic warning
- return animations

That made the bug harder to reason about than the normal finish path.

#RuntimeConfidence

Static checks passed, but the final visual fix still needs manual runtime confirmation on the device/emulator.

The timer animation issue is visual and interaction-dependent, so TypeScript and lint can confirm that the code is valid, but they cannot prove that the red overlay is gone in the actual UI.

#CurrentState

The current unpushed commit has the timer integrated into the task flow.

The app now supports:

- starting a sprint from task details
- opening the timer as `‘/task/timer‘`
- loading the selected task into the sprint screen
- saving active sprint state locally
- restoring active sprint progress from stored end time
- cancelling and clearing active sprint state
- centered headers across the main screens
- updated Expo image assets

The remaining runtime risk is the red hold-to-cancel overlay. The newest implementation hides the overlay with explicit `‘timerOverlayVisible‘` state after manual cancel, but this still needs to be verified by pressing through the cancel flow in the app.

#Verification

Static checks were run after the timer changes:

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

```
““text
npm run lint
exited successfully with one existing warning
““
```

The lint warning is unrelated to today’s timer work:

```
““text
app/(tabs)/subjects.tsx
React Hook useCallback has a missing dependency: ‘GetSubjects’
““
```

The summary above is based on the unpushed commit diff from:

```
““text
origin/timerTask..HEAD
““
```

#FilesChanged

Main timer and task-flow files:

```
““text
app/task/timer.tsx
app/task/_layout.tsx
app/task/viewDetailsTask.tsx
lib/asyncStorage.ts
““
```

Navigation polish files:

```
““text
app/(tabs)/_layout.tsx
app/(tabs)/index.tsx
app/(tabs)/subjects.tsx
app/assignment/upsertAssignment.tsx
app/assignment/viewDetailsAssignment.tsx
app/subject/upsertSubject.tsx
app/subject/viewDetailsSubject.tsx
app/task/upsertTask.tsx
““
```

Image asset files:

```
““text
assets/images/
assets/study-sprint-image-pack/
““
```

New note added:

```
““text
notes/work-report-timer-2026-05-01.md
““
```

#Conclusion

Today’s work moved the timer from an isolated feature into the real task workflow.

The biggest progress was routing and ownership: a sprint now starts from a task, carries that task id into the timer, displays task details during the sprint, and stores active sprint state locally. The surrounding app also received a small consistency pass through centered headers and updated image assets.

The main setback was the manual hold-to-cancel red overlay bug. The implementation has gone through several attempts, and the current version now hides the overlay directly after cancel instead of relying only on moving it out of view. The next step is to verify that final visual behavior live in the app.