

Timer Session Tracking and Dashboard Integration Work Report

#Overview

Today the timer work moved beyond local in-memory behavior and into a more durable sprint-session model.

The main direction was to make sprint time count toward task progress in a safer way, while also surfacing that progress in the app UI. This meant extending the timer flow with database-backed sprint sessions, making task time visible on the task details screen, and continuing the dashboard integration so active or upcoming work is easier to reach.

The work stayed focused on the timer/task/dashboard path rather than broad app refactoring.

#ImplementedFeatures

#SprintSessionPersistence

Moved the timer session model toward a more robust database-backed structure:

- created a 'sprint_sessions' table in Supabase
- added a 'sessionId' field to the local 'ActiveSprint' type in 'lib/asyncStorage.ts'
- updated the timer start flow to create a sprint session in the database before entering the running timer state
- kept local 'active_sprint' storage as the resume handle, but now tied it to a real database session instead of only a task id and end time

This changes the active sprint from being only a local timer state into a recordable session that can later be finalized safely.

#TaskTimeTracking

Added task-level study time tracking:

- added 'totalTimeInSeconds' to the task model in 'lib/types.ts'
- verified that cancelling a sprint updates both 'sprint_sessions' and the task total in the database
- verified that expired sessions also finalize correctly and contribute time as expected

This gives each task a running total of time spent, rather than leaving the timer as a standalone UI action with no durable result on the task itself.

#FinalizeFlowRepair

Adjusted the timer finalize flow so session teardown and restore logic stop fighting each other:

- added a 'finalizeSprintSession(...)' path in 'app/task/timer.tsx'
- updated natural finish, cancel, and expired restore paths to use the database finalize flow
- removed the local active sprint before the finalize RPC completes so the restore effect does not immediately re-open a just-cancelled timer
- added alerts for sprint-session creation/finalization failures instead of silently leaving the screen in a half-running state

This fixed the case where cancelling the timer appeared to work visually, but then the sprint popped back open because restore logic still saw a locally active session.

#TimerStartGuarding

Tightened the sprint-start path in the timer screen:

- delayed 'setIsRunning(true)' until after the 'start_sprint_session' RPC succeeds
- added handling for the returned session id before local sprint state is saved
- added fallback handling for session id shape differences in the RPC response

Before this, the timer UI could enter a partial running state if the database session failed to start, which made the header change without actually starting the timer animation flow.

#TaskDetailsTimeDisplay

Made the recorded task time visible in the task details screen:

- added a local formatter for tracked time in 'app/task/viewDetailsTask.tsx'
- displayed 'Time spent: ...' under the existing metadata block on the task details screen

This is the first direct UI confirmation that the timer is affecting persistent task data rather than only changing temporary timer state.

#DashboardSprintVisibility

Extended the dashboard so it reflects timer/task state more clearly:

- added dashboard support for reading and displaying the current active sprint from local storage
- showed the active sprint task title, description, and remaining time
- added an 'Open Sprint' action that links directly back into the running timer

This gives the user a global way to get back to an already running sprint after navigating away from the timer screen.

#UpcomingDeadlineCards

Added a deadline-based task section to the dashboard:

- added a 'Tasks with upcoming deadlines' section to the dashboard state
- fetched active tasks together with their assignment and subject context
- sorted the tasks by assignment deadline in ascending order
- rendered clickable cards that open the relevant task details screen
- updated the metadata line at the bottom of each card to show subject, assignment, and deadline

This makes the dashboard more useful as a next-action screen instead of only a placeholder when no sprint is running.

#DashboardTaskCompletion

Extended the dashboard task cards so they can directly affect task progress:

- added a 'Mark as completed' action to each upcoming task card
- updated the action to write 'isCompleted = true' back to the matching task row
- reused 'CheckAssignmentCompletion(...)' so assignment completion status stays in sync with task completion
- removed the completed task from the dashboard list immediately after a successful update
- added a confirmation alert before the completion update runs

This gives the dashboard a lightweight task-management action instead of making it read-only.

#DashboardAndSubjectsHelpFlow

Added a small help/info entry point to explain the app structure more clearly:

- added a help button in the dashboard header
- added the same help/info pattern to the subjects screen
- opened a compact modal that explains the app flow as 'Subject -> Assignment -> Task -> Sprint'
- added a clear summary block and a closing action inside the modal

This gives new users a direct explanation of how the app model fits together without leaving the current screen.

#HeaderAndStylingPolish

Continued local UI cleanup around the dashboard and subjects screens:

- aligned the header actions so help and logout use the same general layout pattern on both screens
- converted the dashboard screen from local 'StyleSheet' usage to NativeWind/Tailwind class-based styling
- converted the subjects help/modal block away from 'styles.*' references and into NativeWind classes
- kept the visual structure local to the affected screens rather than introducing shared abstractions

This keeps the dashboard and subjects screens stylistically closer to each other while staying within the current app structure.

#ActiveSprintDashboardFix

Fixed a rendering bug in the dashboard state logic:

- the 'Tasks with upcoming deadlines' section had been placed inside the 'no active sprint' branch
- when a sprint was active, the upcoming task section disappeared entirely
- moved the upcoming task section out of that conditional so both the active sprint card and upcoming tasks render together

This keeps the dashboard useful while a sprint is already running instead of hiding the rest of the user's near-term work.

#ProblemsAndSetbacks

#QuotedColumnNames

The first major issue today came from the new sprint-session SQL functions using unquoted camelCase column names.

The database columns used names such as:

- 'sessionId'
- 'taskId'
- 'startedAt'
- 'elapsedSeconds'

Without quotes, Postgres treated these as lowercase names like 'sessionid' and 'taskid', which caused RPC failures when starting or finalizing sprint sessions.

This had to be corrected in the SQL functions before the app-side timer integration could work.

#RowLevelSecurity

The next blocker was row-level security on 'sprint_sessions'.

Even after the SQL functions matched the correct columns, session creation still failed until the insert/select/update permissions allowed authenticated users to work with their own sprint-session rows.

This was a necessary database-layer fix before the new robust timer flow could be tested end to end.

#CancelRestoreRace

Another significant bug showed up after the new finalize flow was wired in:

- the cancel animation ran
- the timer visually closed
- then the sprint reopened immediately

The cause was that the restore effect still found 'active_sprint' in local storage while the cancel/finalize path was still finishing. Removing the local active sprint earlier in the finalize path fixed that race.

#DashboardListInterpretation

There was also a dashboard-listing issue where the upcoming-deadlines section could appear to only show tasks from one subject.

The actual cause was not the subject join itself, but the fact that the list had been truncated after sorting. That made the section biased toward whichever subject owned the earliest deadlines in the current data.

#UpcomingTasksVisibilityBug

Another dashboard bug appeared after the active sprint card had been added.

The issue was not in the Supabase query itself. The problem was that the upcoming-deadlines section was rendered only in the 'no active sprint' branch of the dashboard conditional.

This meant:

- the active sprint card appeared correctly
- the upcoming task data was still loaded
- but the list was hidden whenever a sprint existed

The fix was to move the upcoming task section outside that conditional so the dashboard can show both at once.

#CurrentState

The timer/task flow now goes further than yesterday's integration work.

The app now supports:

- creating a real sprint session in the database when a timer starts
- finalizing sprint sessions on cancel and expiry
- adding tracked session time into 'tasks.totalTimeInSeconds'
- showing tracked task time on the task details screen
- reopening the active sprint from the dashboard
- showing upcoming deadline task cards even while a sprint is active
- marking upcoming dashboard tasks as completed with a confirmation step
- opening a help/info modal from both the dashboard and subjects headers
- using NativeWind/Tailwind styling for the dashboard screen and the subjects help modal block

At this point, the timer is no longer only integrated into the task route. It is now also contributing durable progress data back into the task model and exposing more of that state in surrounding screens.

#Verification

During today's work, the following behaviors were verified manually through the app plus database inspection:

- sprint creation now succeeds after fixing quoted column names and RLS
- cancelling a sprint updates both 'sprint_sessions' and 'tasks.totalTimeInSeconds'
- expired sprint finalization also updates the database as expected
- the cancel flow no longer reopens the timer immediately after the close animation

Static checks were also run during the implementation work:

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

```
““text
npm run lint -- app/task/timer.tsx
exited successfully
““
```

```
““text
npm run lint -- app/task/viewDetailsTask.tsx
exited successfully
““
```

```
““text
npm run lint -- app/(tabs)/index.tsx
exited successfully
““
```

```
““text
npx eslint app/(tabs)/index.tsx
exited successfully
““
```

```
““text
npx eslint app/(tabs)/subjects.tsx
completed with one existing warning
““
```

The later dashboard/subjects polish work was verified with local static checks and code inspection. The summary above is based on today's working-tree changes plus the live runtime/database checks done while fixing the timer session flow.

#FilesChanged

Main timer/session files:

```
```text
app/task/timer.tsx
lib/asyncStorage.ts
lib/types.ts
```
```

Task details and dashboard files:

```
```text
app/task/viewDetailsTask.tsx
app/(tabs)/index.tsx
app/(tabs)/subjects.tsx
lib/progress.ts
```
```

New note added:

```
```text
notes/work-report-timer-2026-05-02.md
```
```

#Conclusion

Today's work turned the timer into something closer to a real task-tracking feature instead of only a screen-local countdown.

The biggest progress was introducing sprint sessions as database-backed records, finalizing them into tracked task time, and then surfacing that state back into the app through task details and the dashboard. The later dashboard follow-up made that surrounding UI more useful by keeping upcoming tasks visible during active sprints, allowing quick task completion from the dashboard itself, and adding a lightweight in-app explanation of the subject/assignment/task/sprint model.