PROJECT REPORT: EduTimer - A FOCUS & PRODUTIIVITY APP

SUBBMTTED BY:

Aisha Khan (22SW017)

Zaher Ahned (22SW077)

SUBBMTTED TO:

Ma'am Mariam Memon

PROJECT:

EduTimer (CEP for Mobile Application Development)

Table of Contents

1. Real-World Problem Identification

In today's digital world, students and professionals face a constant battle for focus. The endless stream of social media notifications, emails, and other digital distractions makes it incredibly difficult to dedicate deep, focused time to important tasks. This leads to poor productivity, increased stress, and often, last-minute, low-quality work.

The core problem isn't a lack of desire to work; it's the lack of a structured tool to help manage focus and build a productive habit.

2. Proposed Solution

EduTimer is a mobile application designed to solve this exact problem. It's built on the **Pomodoro Technique**, a proven time-management method.

The app provides a simple, engaging, and customizable tool that helps users structure their work. The core idea is to break down work into intervals:

1. **Focus:** A 25-minute (customizable) session of pure, uninterrupted focus on a single task.
2. **Break:** A 5-minute (customizable) break to relax and recharge.

By adding features like task management (so users know *what* to focus on), progress tracking (to see their hard work pay off), and background audio (to create a focused environment), EduTimer acts as a personal productivity coach in the user's pocket.

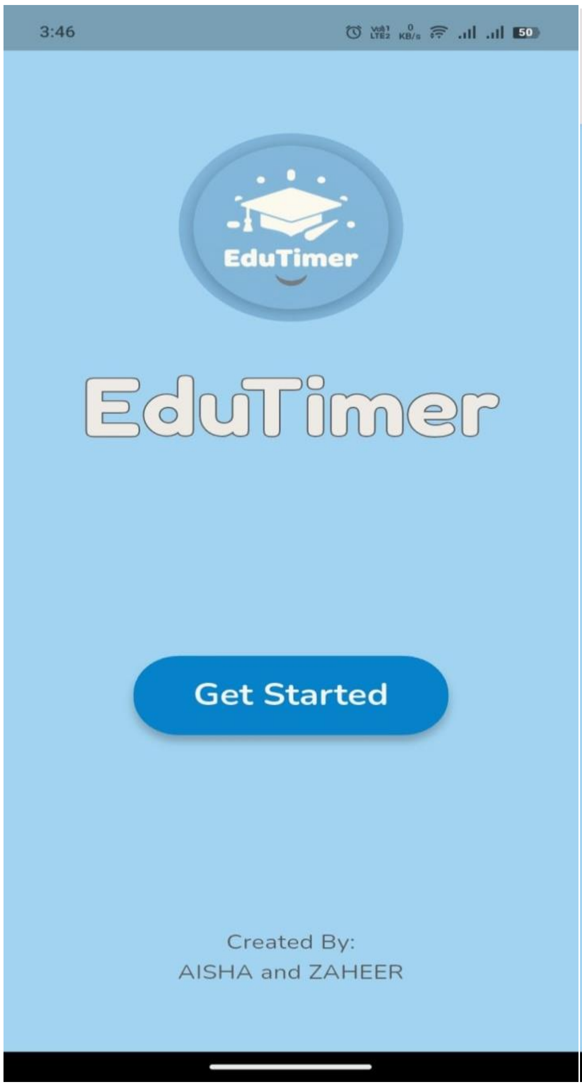
We chose **Flutter** as our development framework because it allows us to build a high-quality, beautiful application for both Android and iOS from a single codebase, directly addressing the project's cross-platform requirements.

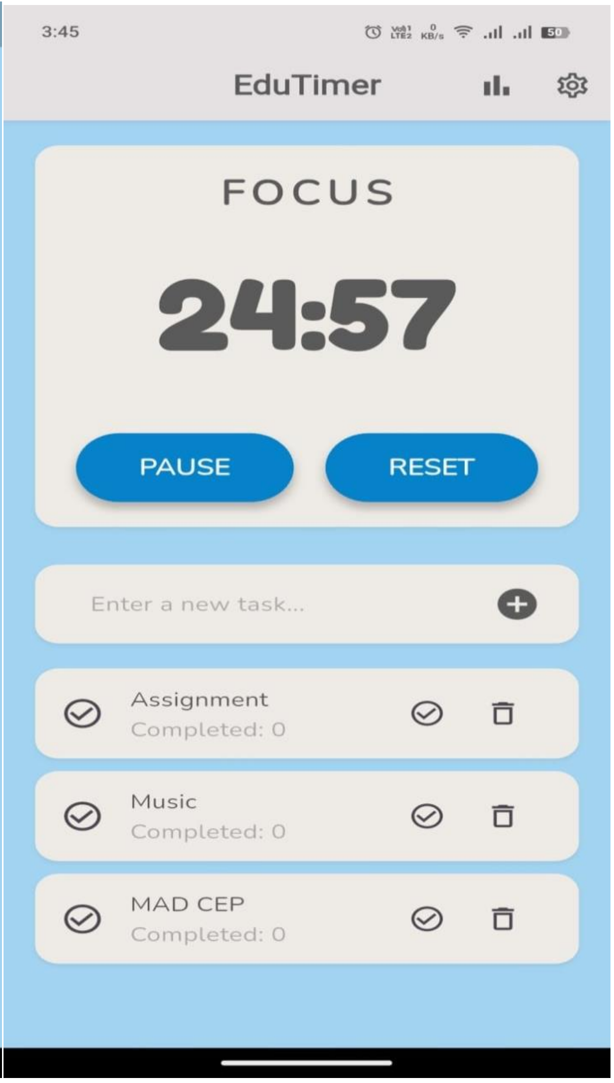
3. Responsive User Interfaces

A key requirement was that the app must work uniformly on all screen sizes. We achieved this through several methods:

- Flexible Layouts: We used Flutter's flexible widgets like Column, Row, and Card. This ensures that the elements on the screen can grow or shrink gracefully.
- Handling the Keyboard: A major responsiveness challenge is the on-screen keyboard. When the user taps the "Enter a new task" field, the keyboard appears and reduces the available screen space.
  - **Solution:** We wrapped our entire Pomodoro Screen inside a SingleChildScrollView. This widget is smart; it allows the user to scroll the screen up when the keyboard appears, preventing the dreaded "Bottom Overflow" error and ensuring the app is usable on all screen sizes, big or small.

SCREENSHOTS OF THE APP :

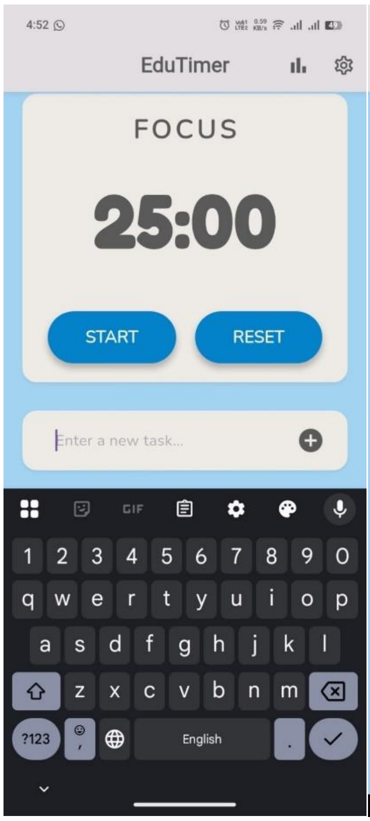


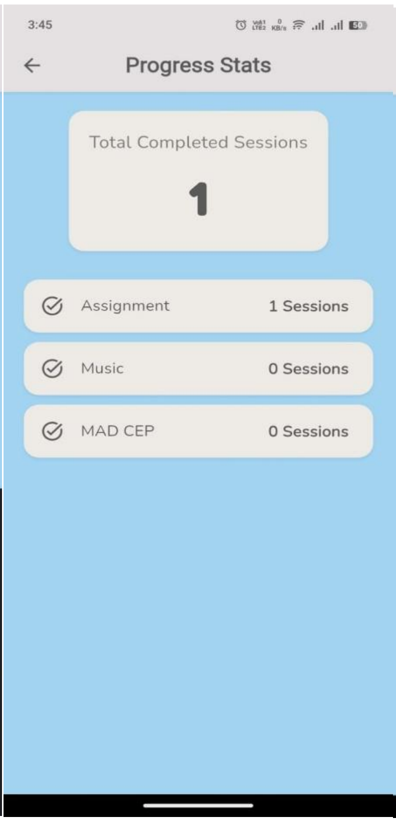


Assignment
Completed: 0

Music
Completed: 0

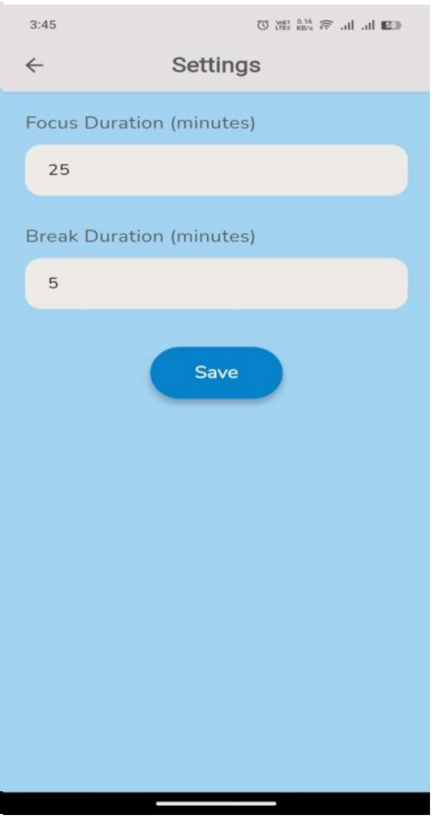
MAD CEP
Completed: 0





Total Completed Sessions
1

Assignment	1 Sessions
Music	0 Sessions
MAD CEP	0 Sessions



Focus Duration (minutes)
25

Break Duration (minutes)
5

4. Data Storage

Our app needed to remember user data even if the app was closed. We used two different *local* storage methods, based on the type of data.

Justification for Local Storage:

We chose local storage (on the phone) instead of a cloud database because EduTimer is a personal, single-user app. A cloud database would add unnecessary complexity, cost, and require a constant internet connection. Local storage is faster, more reliable for this task, and respects user privacy.

1. For User Settings (Focus/Break Time):

- **Database Used:** shared_preferences
- **Justification:** This package is perfect for storing simple key-value pairs (e.g., focus_duration = 30). It's extremely lightweight and fast. Using a full SQL database for just two numbers would be inefficient.

2. For Task Lists & Progress Stats:

- **Database Used:** sqflite
- **Justification:** This is a full-fledged SQL database on the phone. We needed it for our *structured* data. Each "Task" has an ID, a Title, and a pomodoroCount (number of completed sessions). sqflite allows us to easily add, delete, and update specific tasks (like increasing the session count when a user presses the 'check' button).

5. APIs, Packages, and Plug-ins Used

We used several third-party packages to add complex features quickly and follow coding standards (by not "reinventing the wheel").

google_fonts:

- **Justification:** To create a beautiful and consistent UI. We used 'Fredoka' and 'Nunito' fonts to match our modern, friendly theme, which isn't possible with default system fonts.

flutter_local_notifications:

- **Justification:** This was critical for the app's core function. It allows the app to send an alert ("Time for a break!") even when the app is in the background or the phone is locked.

audioplayers:

- **Justification:** To improve the user's focus. This package allows us to play calming background music during focus sessions and relaxing sounds during breaks, making the experience more immersive.

path:

- **Justification:** A helper package for sqflite. It finds the correct, platform-specific folder on the phone to safely store the database file.

6. Issues and Bugs Encountered (and Resolved)

This was the most challenging part of the project. We faced numerous "real-world" problems that weren't simple coding errors.

Bug 1: The "Everything is an Error" Bug

- **Problem:** After creating our first PomodoroScreen file, the code was filled with errors like StatefulWidget not found, Text not found, and Scaffold not found.
- **Resolution:** We realized we had forgotten the most basic line: import 'package:flutter/material.dart';. This taught us that all basic Flutter widgets live inside this core library.

Bug 2: The "Infinite Loading Spinner" on Web

- **Problem:** The app worked perfectly on Chrome. As soon as we added the sqflite (database) package, the app would launch but forever show a loading spinner.
- **Resolution:** After research, we learned that sqflite is a *native* package. It talks directly to the phone's (Android/iOS) hardware. It **cannot run on a web browser**. This forced us to change our entire testing process from Chrome to a real Android device.

Bug 3: The "Unsupported class file major version 68" Error (The Gradle/Java War)

- **Problem:** When we tried to run the app on an Android phone, the build failed with this confusing Gradle error.
- **Explanation:** This was a deep-level environment issue. The Java version installed on our computer (e.g., Java 24) was too new for the project's default Gradle (Android's build system) version.
- **Resolution:** This took several steps.

1. First, we upgraded the project's Gradle version in gradle-wrapper.properties and build.gradle.kts files.
2. This wasn't enough. The final fix was to edit the android/gradle.properties file and add a line (org.gradle.java.home=...) that *forced* Gradle to ignore the system's new Java and use a specific, compatible version (Java 17) that we downloaded.

Bug 4: The "Space in Path" Error

- **Problem:** After fixing the Java issue, the build *still* failed with a new error: Failed to create parent directory 'C:\Users\Zaheer'.
- **Explanation:** Our project was located at C:\Users\Zaheer Ahmed\.... The Android build tools cannot handle the **space** in the "Zaheer Ahmed" folder name.
- **Resolution:** We moved the entire project folder to a "safe" path: C:\projects\studytimer_app. This instantly fixed all path-related build errors.

Bug 5: The "Bottom Overflowed" Error (The Keyboard Bug)

- **Problem:** The app ran, but when we tapped the "Enter a new task" field, the keyboard popped up and caused a scary yellow-and-black striped "BOTTOM OVERFLOWED" error.
- **Explanation:** The keyboard reduced the screen's height, and there wasn't enough space to show the timer, buttons, and task list.
- **Resolution:** We wrapped the main Column in pomodoro_screen.dart with a SingleChildScrollView widget. This instantly fixed the bug by making the screen scrollable when the keyboard is open.

Bug 6: The CardThemeData Error

- **Problem:** While implementing our new light theme, we got an error: The argument type 'CardTheme' can't be assigned to the parameter type 'CardThemeData?'.
- **Explanation:** We were using an outdated class name (CardTheme).
- **Resolution:** We corrected the code in app_theme.dart to use the proper CardThemeData class, which fixed the theme.

Project Constraints (How We Met Them)

- **Constraint 1 (Platforms & Screen Sizes):** Met by using **Flutter** (for cross-platform) and **SingleChildScrollView / flexible widgets** (for screen sizes), as described in Section 3.
- **Constraint 2 & 3 (Coding Standards & Maintainability):** Met by:

1. **Separation of Concerns:** We didn't put all code in one file. We created separate "service" files (database_helper.dart, notification_service.dart, audio_service.dart) and separate files for each screen.
2. **Central Theme:** Using app_theme.dart means we can change the entire app's look from one file. This is highly maintainable.

- **Constraint 4 (Data Storage):** Met by using **sqflite** and **shared_preferences**, as justified in Section 4.
- **Constraint 5 (Optimal State Management):** Met by using **setState**.

  - **Justification:** For an app of this size, setState is the most optimal and maintainable choice. Using complex state management tools (like Riverpod or BLoC) would be "over-engineering." setState is built-in, fast, and easy to understand, which perfectly matches our goal of simple, maintainable code.