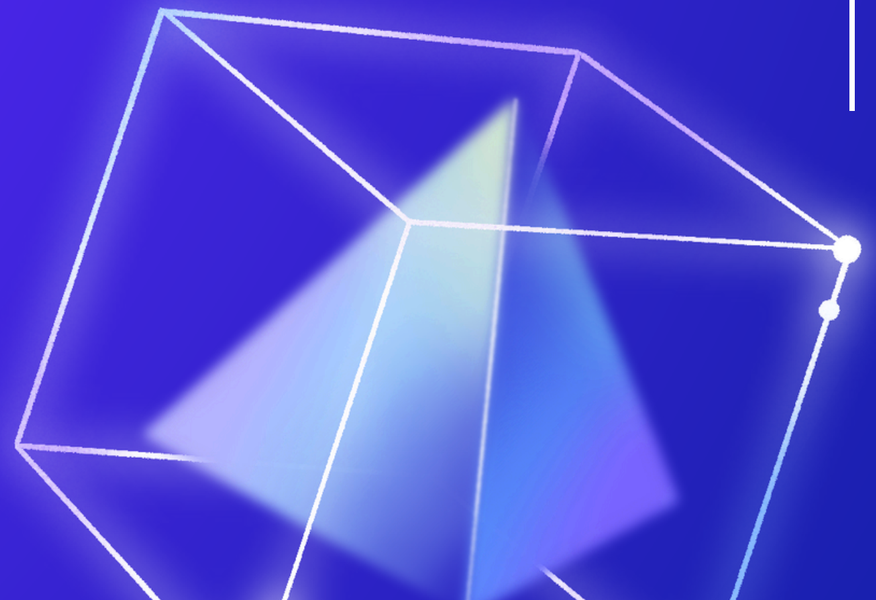
MAINTANACE OF

ACHIVA

G4

TABLE OF CONTENTS



INTRODUCTION

Project Overview

Achiva is a goal-tracking and productivity platform designed to support individuals and teams in setting, tracking, and achieving their goals. Our maintenance project focused on improving its usability, performance, and long-term adaptability.

This phase builds upon the original version developed in SWE 444, applying real-world software maintenance practices to evolve the system based on user needs and technical insights.

SYSTEM BACKGROUND

Achiva supports users with features like personalized goal planning, task reminders, AI chat assistance, and a social productivity leaderboard.

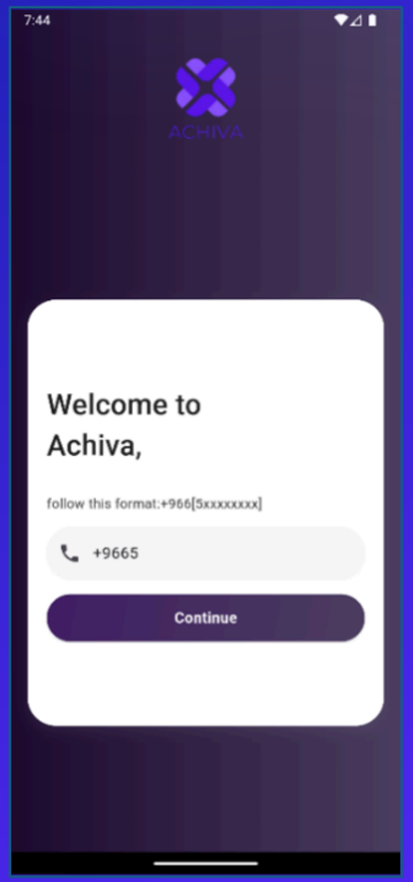
Who Uses It?

- Students and professionals
- Project teams
- Mentors and coaches

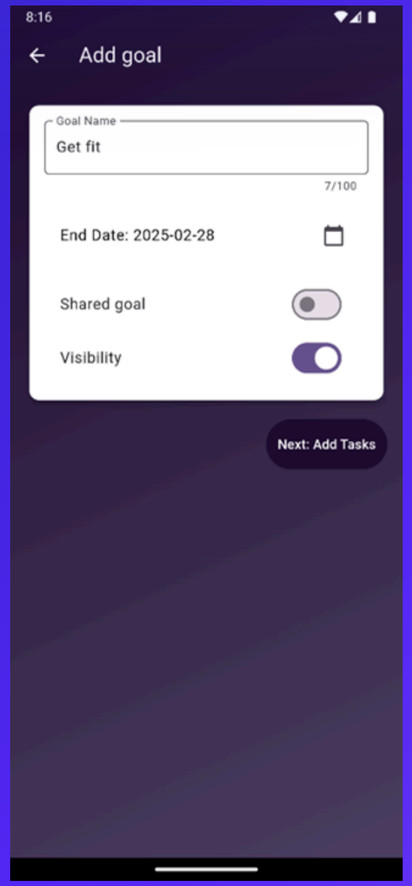
Main Features

AI-powered insights, collaborative goals, shared tasks, reminders, and visual productivity feedback.

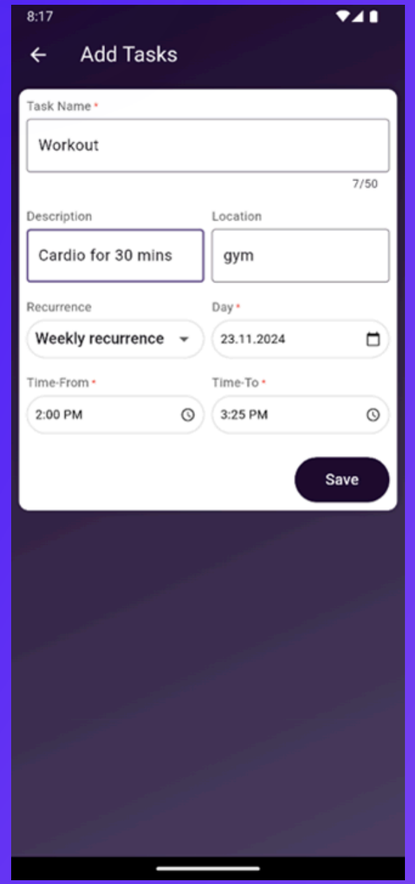
APP SCREENSHOTS



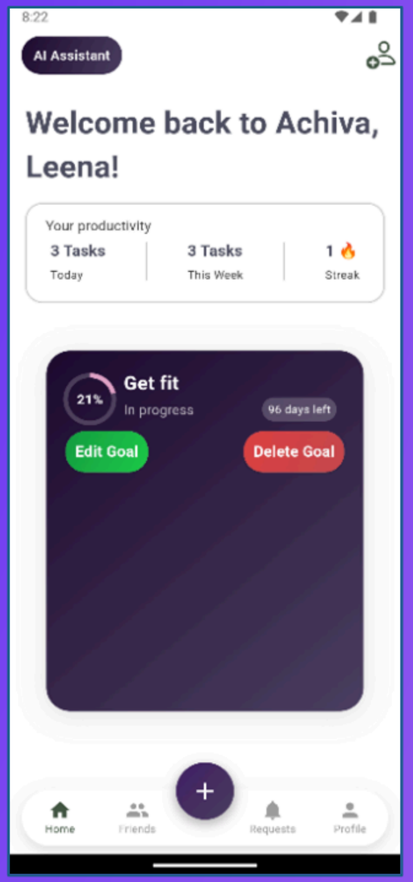
SignUp page



Add goal page

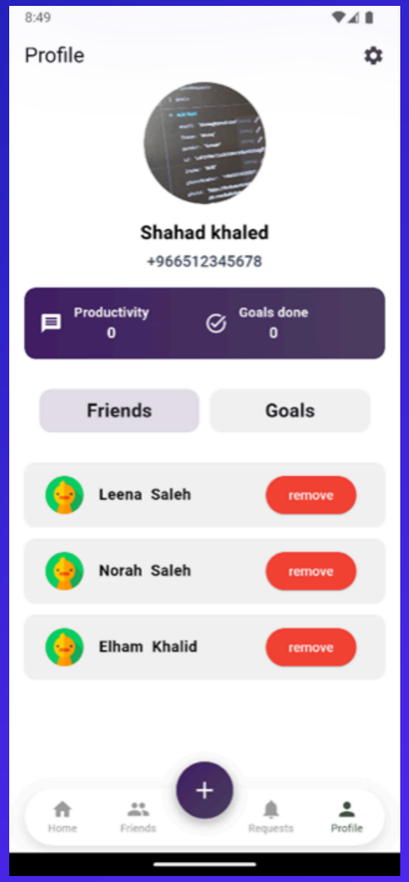


Add taskpage

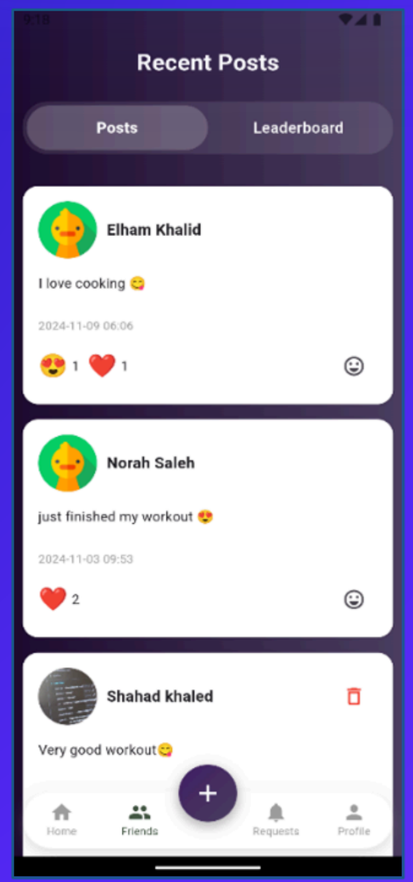


Home page

APP SCREENSHOTS

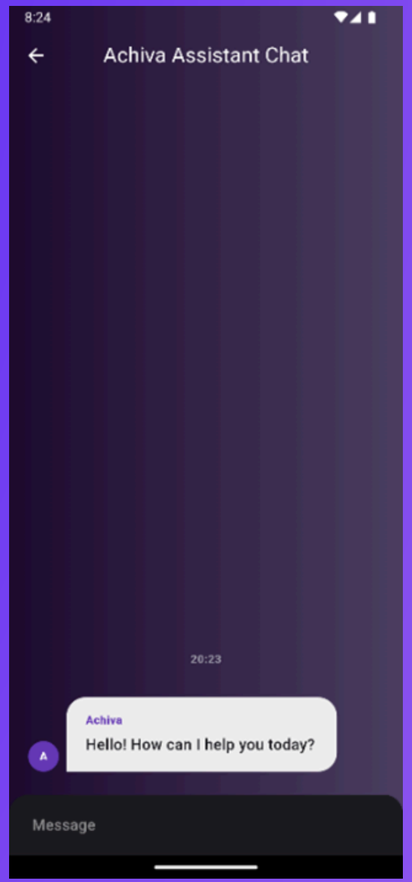


Profile page



Posts page

Leaderboard page



AI chat page

PROGRAM COMPREHENSION

Understanding the system

We began by reviewing existing documentation from the previous course, exploring the codebase, and inspecting the system's features directly through the app. This helped us build a mental model of how components interact and where key functionalities are implemented. We also studied the database schema and how tasks and goals are structured and stored.

PROGRAM COMPREHENSION

Types of Comprehension Used

- Use Case Modeling

We created a detailed Use Case Diagram to map user interactions with Achiva. This made it easier to understand feature relationships and prioritize areas for maintenance.

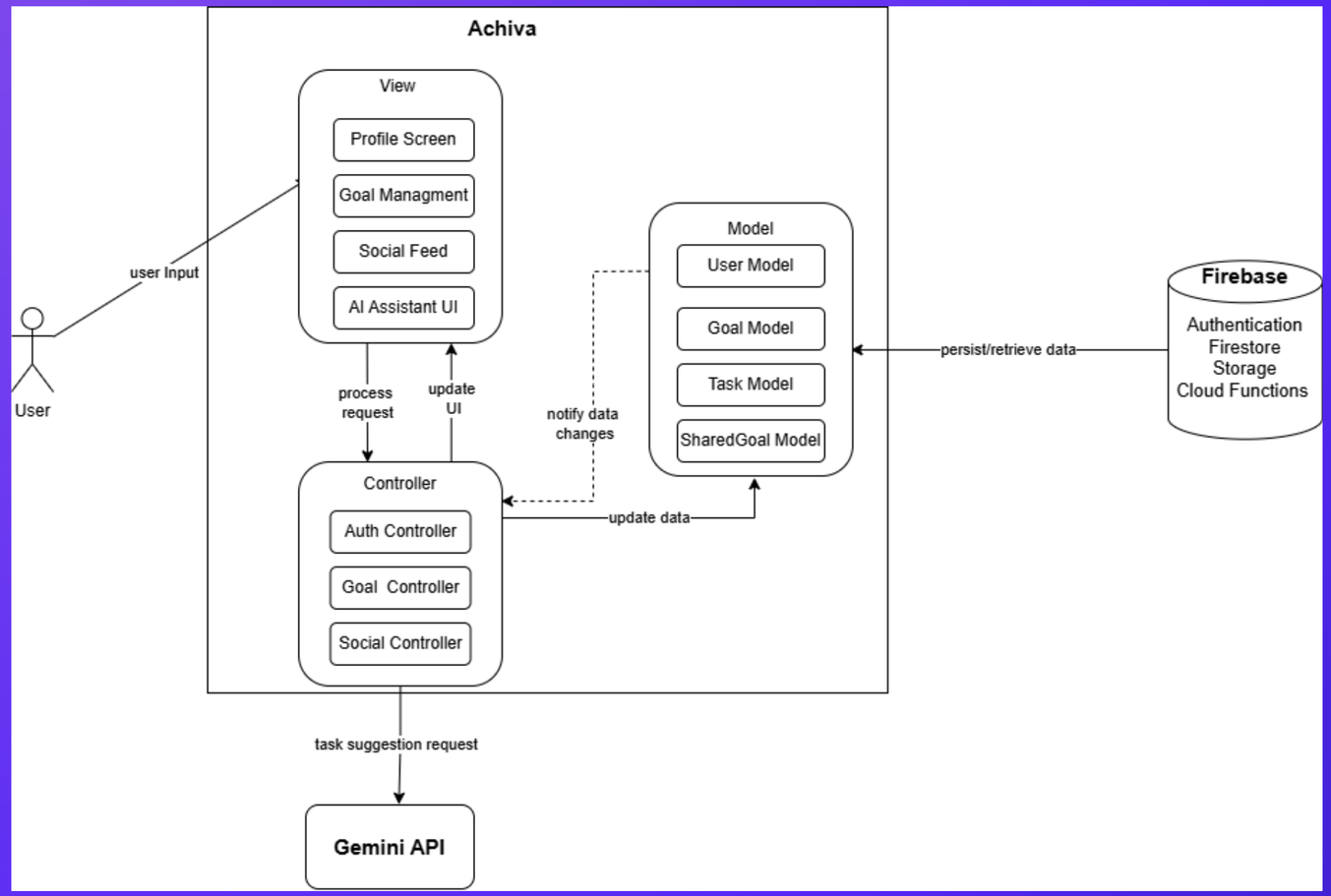
Use Case Diagram

PROGRAM COMPREHENSION

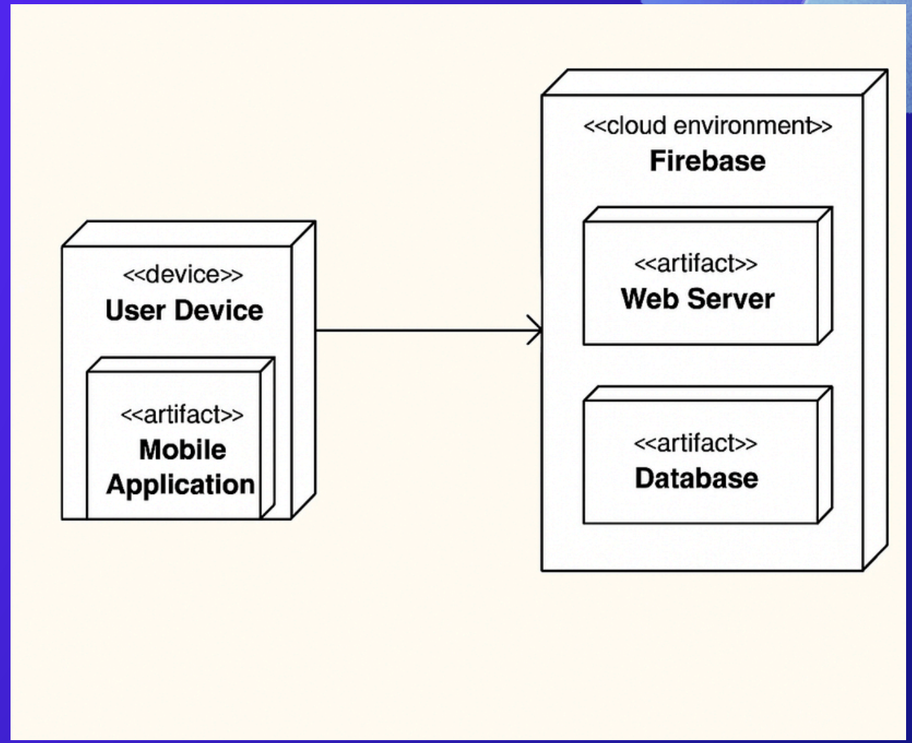
Types of Comprehension Used

- Architecture & Deployment Analysis

By analyzing the MVC architecture, we identified how the frontend, backend, and Firebase database are connected. We then visualized this structure with a Deployment Diagram to clarify system distribution and hosting.



System Architecture Diagram



Deployment Diagram

MAINTENANCE REQUEST (MR) MANAGEMENT

01

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin varius, eros nec efficitur euismod, lectus turpis sollicitudin augue, non ultricies enim nunc sit amet augue.

02

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin varius, eros nec efficitur euismod, lectus turpis sollicitudin augue, non ultricies enim nunc sit amet augue.

03

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin varius, eros nec efficitur euismod, lectus turpis sollicitudin augue, non ultricies enim nunc sit amet augue.

04

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin varius, eros nec efficitur euismod, lectus turpis sollicitudin augue, non ultricies enim nunc sit amet augue.

MAINTENANCE REQUEST (MR)

PLANNED MAINTENANCE ACTIVITIES

During this project, we followed a structured maintenance plan to manage and prioritize requests effectively.
We used:

- A centralized request system to track changes from submission to release.
- Bi-weekly review meetings to evaluate new requests with the team.
- Cost and resource estimation for each request to ensure feasibility.
- Version control (GitHub) with feature branches and code reviews to manage changes.

MR_1:TASK ASSIGNMENT IN SHARED GOALS

MR_2:TASK DUE DATE NOTIFICATIONS ENHANCEMENT

FROM THIS MR MANAGEMENT PROCESS, WE LEARNED THAT:

01

- Structuring the process improves tracking and transparency

02

- Prioritization helps us focus on features with the highest user impact

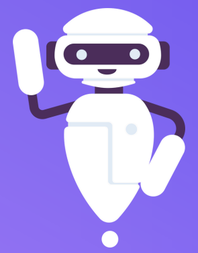
03

- Detailed impact analysis reduces the risk of system failures

04

- Documentation and training ensure smooth user adoption

SLICING

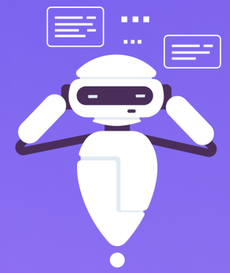


OBJECTIVE 01

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam vel euismod ipsum. Proin fermentum dolor vel est fermentum, at sagittis diam.

OBJECTIVE 02

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam vel euismod ipsum. Proin fermentum dolor vel est fermentum, at sagittis diam.



OBJECTIVE 03

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam vel euismod ipsum. Proin fermentum dolor vel est fermentum, at sagittis diam.

REFACTORING

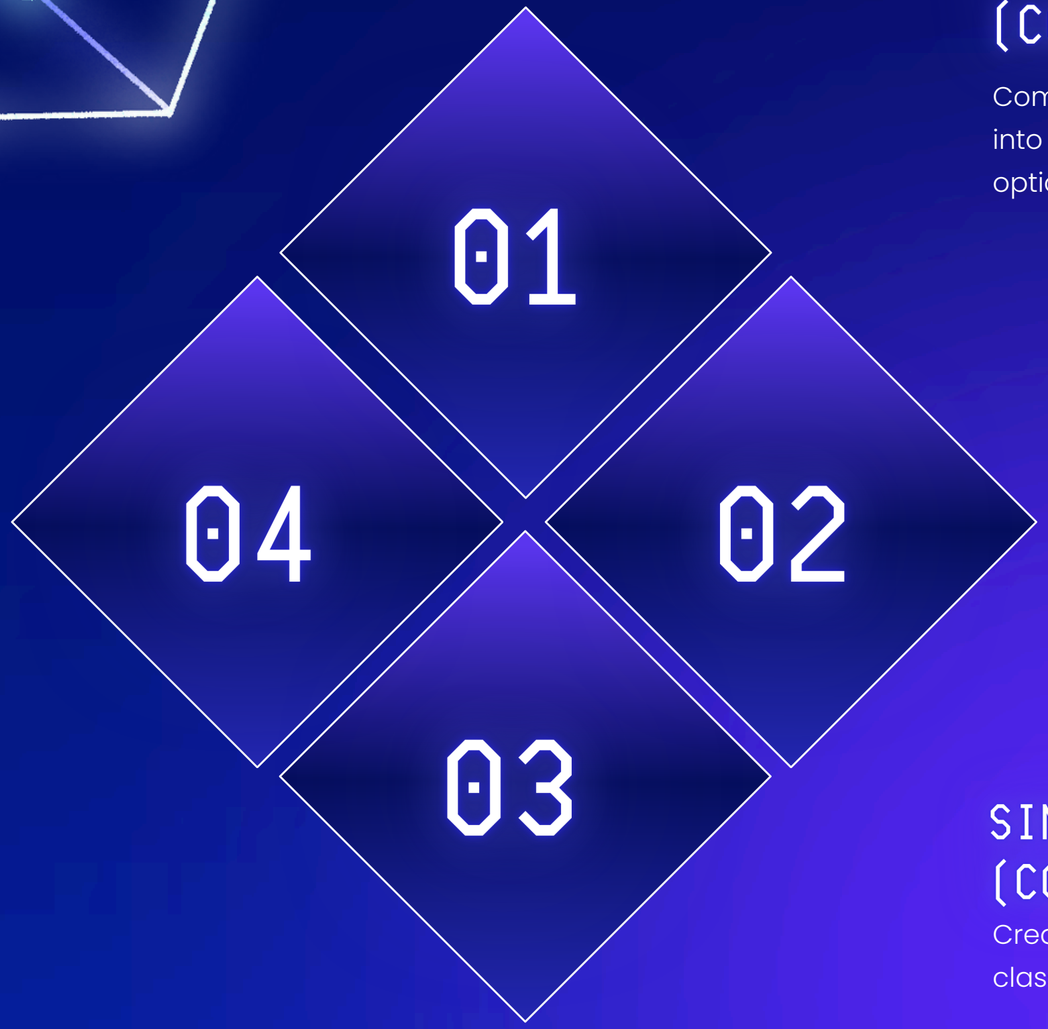
CODE SMELLS IDENTIFIED

the add task feature had **2 classes**

1. add_task_independitly: adding a task to a previously added goal
2. add_task: adding a task for a new goal

code smells found :

1. Duplicated Code.
2. Large Class.
3. Long Method. (_createGoalAndAddTask)
4. Feature Envy.
5. Inconsistent Validation Logic.



MERGE DUPLICATE CLASSES (COLLAPSE HIERARCHY)

Combine AddTaskPage and AddTaskIndependentlyPage into a single class with configuration options(isIndependent: bool).

EXTRACT FIRESTORE OPERATIONS (EXTRACT CLASS)

Create a dedicated TaskRepository class to handle all Firestore operations

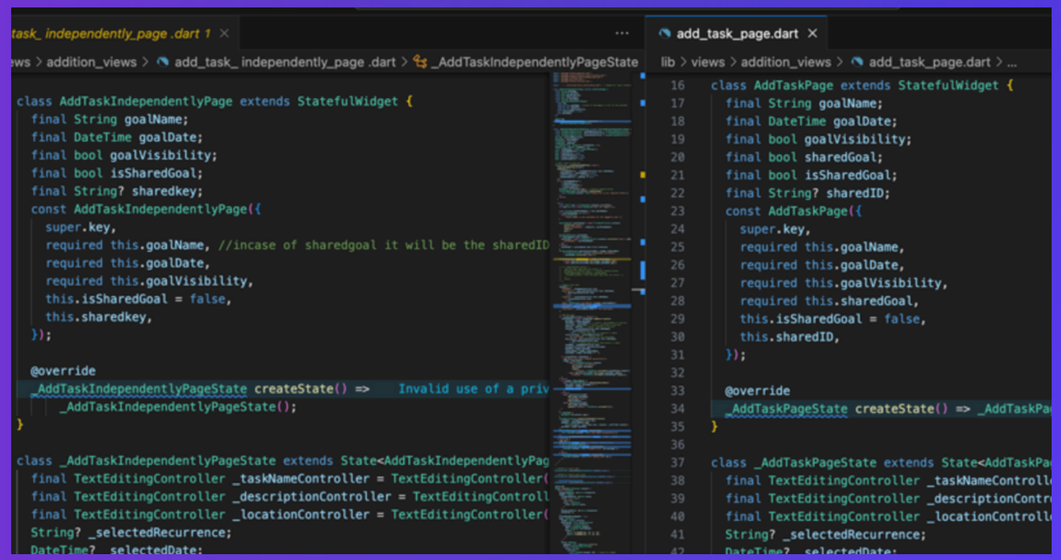
CREATE DATETIME HELPER (EXTRACT METHOD)

Extract date/time operations into a dedicated helper class.

SIMPLIFY VALIDATION (CONSOLIDATE CONDITIONALS)

Create a unified validation system using a TaskValidator class

MERGE DUPLICATE CLASSES (COLLAPSE HIERARCHY)



BEFORE

```dart
class AddTaskPage extends StatefulWidget {
  final String goalName;
  final DateTime goalDate;
  final bool goalVisibility;
  final bool isSharedGoal;
  final String? sharedKey;
  final bool isIndependent;

  const AddTaskPage({
    super.key,
    required this.goalName,
    required this.goalDate,
    required this.goalVisibility,
    this.isSharedGoal = false,
    this.sharedKey,
    this.isIndependent = false,
  });

  @override
  _AddTaskPageState createState() => _AddTaskPageState();    Invalid use of a private type in a public A
}
```

AFTER

EXTRACT FIRESTORE OPERATIONS (EXTRACT CLASS)

```
    if (createdTasks.isNotEmpty) {
      log("Recurring tasks created successfully");
      widget.isSharedGoal
          ? await FirebaseFirestore.instance
              .collection('sharedGoal')
              .doc(widget.sharedID)
              .update(
                  {'notasks': FieldValue.increment(createdTasks.length)})
          : await goalsCollectionRef.doc(widget.goalName).update(
              {'notasks': FieldValue.increment(createdTasks.length)});
    }
  } else {
    if (widget.sharedGoal) {
      await SharedGoalManager().addTaskToSharedGoal(
        sharedID: widget.sharedID ?? "",
        taskData: taskData,
        context: context,
      );
    } else {
      // for non-shared goals
      await goalsCollectionRef
          .doc(widget.goalName)
          .collection('tasks')
          .add(taskData);
    }
```

BEFORE

```
53
54        if (createdTasks.isNotEmpty) {
55          log("Recurring tasks created successfully");
56          widget.isSharedGoal
57              ? await _taskRepository.updateTaskCount(
58                  goalName: widget.goalName,
59                  increment: createdTasks.length,
60                  isShared: widget.isSharedGoal,
61                  sharedKey: widget.sharedKey,
62                )
63              : await goalsCollectionRef.doc(widget.goalName).update(
64                  {'notasks': FieldValue.increment(createdTasks.length)});
65        }
66      } else {
67        if (widget.isSharedGoal) {
68          await SharedGoalManager().addTaskToSharedGoal(
69            sharedID: widget.sharedKey ?? "",
70            taskData: taskData,
71            context: context,
72          );
73        } else {
74          await _taskRepository.addTask(
75            goalName: widget.goalName,
76            taskData: taskData,
77            isShared: widget.isSharedGoal,
78            sharedKey: widget.sharedKey,
79          );
```

AFTER

- **getUserDocument():** it will get the user collection document
- **createGoal():** will create a goal and handle if the goal is a shared goal or not when the isIndependent is false
- **addTask():** will add a task and handle if the task is for a shared goal or not

EXTRACT FORM VALIDATION (EXTRACT METHOD)

```
71    }
72
73    Future<void> updateUserData() async {
74      try {
75        // Validate input fields
76        if (_fnameController.text.isEmpty ||
77            _lnameController.text.isEmpty ||
78            _emailController.text.isEmpty ||
79            widget.layoutCubit.chosenGender == null) {
80          showSnackBarWidget(
81            message: "Please fill in all fields",
82            successOrNot: false,
83            context: context
84          );
85          return;
86        }
87
88        // Check if email is already in use by another user
89        if (await checkIfEmailExists(_emailController.text) &&
90            _emailController.text != widget.layoutCubit.user!.email) {
```

BEFORE

```
65    bool isFormValid(BuildContext context) {
66      if (_fnameController.text.isEmpty ||
67          _lnameController.text.isEmpty ||
68          _emailController.text.isEmpty ||
69          widget.layoutCubit.chosenGender == null) {
70        showSnackBarWidget(
71          message: "Please fill in all fields",
72          successOrNot: false,
73          context: context,
74        );
75        return false;
76      }
77      if (!isValidEmail(_emailController.text)) {
78        showSnackBarWidget(
79          message: "Please enter a valid email address",
80          successOrNot: false,
81          context: context,
82        );
83        return false;
84      }
85      return true;
86    }
```

AFTER

IMPROVEMENTS ACHIEVED

1. Enhanced Code Organization
Merged Duplicate Classes has improved readbility, The new TaskRepository class making future changes to data access easier to implement.

2. Improved Readability
Simplified Validation Logic: validation rules explicit and easy to understand.
Better Separation of Concerns: UI logic is now separate from business logic and database operations

3. Easier Maintenance
Single Point of Modification
Reduced Method Length: made the code more manageable.
Type Safety Improvements: DateTime operations now benefit from stronger typing, reducing potential runtime errors.

4. Optimized Database Access

5. Resource Efficiency
Memory Usage: fewer duplicate objects in memory during task creation.

6. Reduced Code Size

THANK YOU!