A project

ELEGANT & FUNCTIONAL TODO TASK TRACKER APP

CSE 400: Software Development Project **IV**

SUBMITTED BY

Name	ID	Intake
Mohammad Hasibul Hasan	21225103319	49/8
Isme Azam Sakib	21225103320	49/8
Shohayel Ahmed	21225103322	49/8
Tasabil Islam Mojumder	21225103353	49/8
Imam Hossain	21225103354	49/8

SUPERVISED BY:

Bijon Mallik
Lecturer,
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY (BUBT)

12th, January, 2018

Abstract

This project presents the development of a secure, cloud-based To-Do application designed using **Flutter** and integrated with **Firebase** services for authentication and real-time database functionality. The application allows users to create, update, delete, and track personal tasks through a user-friendly mobile interface. Key features include **secure user authentication**, **task management**, and **cloud storage with Firestore**, ensuring data is synchronized across devices.

The main motivation behind this project is the growing demand for productivity tools that are simple, secure, and accessible from anywhere. Existing task apps often come with limitations such as paid subscriptions, limited offline support, or lack of user data security. Our app addresses these issues by offering a lightweight yet secure alternative that functions entirely on open-source and free tools, making it ideal for students and professionals alike.

The system was developed following the **Agile methodology**, which enabled step-by-step feature integration and frequent testing. Throughout development, tools like Android Studio, Visual Studio Code, and Firebase Console were used. The final product delivers a practical and scalable solution for daily task tracking, laying the groundwork for future expansion such as reminders, collaboration features, and cross-platform deployment.

List of Figures

Table of Contents

Chapter 1
INTRODUCTION

1.1. Problem Specification

Most task management apps available today are either overly complex, restricted by paywalls, or lack basic security and synchronization features. Many don't provide secure user authentication or real-time cloud storage. This makes it hard for users to manage their tasks across multiple devices securely. Our project aims to solve this by creating a simple, secure, and reliable To-Do app that supports user login, task management, and cloud syncing via Firebase.

1.2. Objectives

- To develop a mobile-friendly To-Do application using Flutter.
- To implement secure user authentication using Firebase Auth.
- To enable task creation, editing, deletion, and completion tracking.
- To store and sync tasks using Firebase Firestore.
- To design a clean and intuitive UI with onboarding support.

1.3. Scope

- Requirement gathering and feature planning.
- Setting up the Flutter environment.
- Firebase project configuration (authentication and Firestore).
- Implementing authentication (sign-up, login, logout).
- Building UI screens (onboarding, dashboard, task creation, task details).
- Integrating Firestore for real-time data handling.
- Testing, debugging, and final deployment.

1.4. Organization Of Project Report

Chapter 1: Introduction to the problem, objectives, and report flow.

Chapter 2: Background research and analysis of similar systems.

Chapter 3: Tools, models, and system design diagrams.

Chapter 4: Implementation details and core development outcomes.

Chapter 5: Testing, evaluation, and final deployment insights.

Chapter 6: Conclusion, limitations, and future enhancements.

Chapter 2
BACKGROUND

2.1. Existing System Analysis

We reviewed multiple to-do apps like **Todoist**, **Google Tasks**, and **Microsoft To Do**:

- **Todoist** is feature-rich but requires a premium plan for full functionality.
- **Google Tasks** is simple but lacks collaboration and advanced sorting.
- **Microsoft To Do** is limited to Microsoft's ecosystem and lacks flexibility.
  These systems inspired our approach to offer essential features without unnecessary complexity or paid features.

2.2. Supporting Literatures

- Firebase Documentation: Auth & Firestore official guides.
- Google's Flutter documentation for UI and state management.
- PCMag and CRM.org reviews on existing to-do list apps.
- YouTube video tutorials explaining Firebase integration with Flutter.

These sources supported our understanding of best practices in secure app development, authentication systems, and mobile UI/UX.

Chapter 3
SYSTEM ANALYSIS & DESIGN

3.1. Technology & Tools

- **Frontend:** Flutter (Dart language)
- **Backend:** Firebase Authentication & Firestore
- **IDE:** Android Studio, VS Code
- **Version Control:** Git + GitHub
- **Platforms:** Android (and optionally iOS later)

3.2. Model & Diagram

3.2.1. Model (SDLC/Agile/Waterfall/OOM)

We followed the **Agile model**, allowing us to build and test in iterations. This gave flexibility to handle feedback, fix bugs quickly, and develop feature by feature.

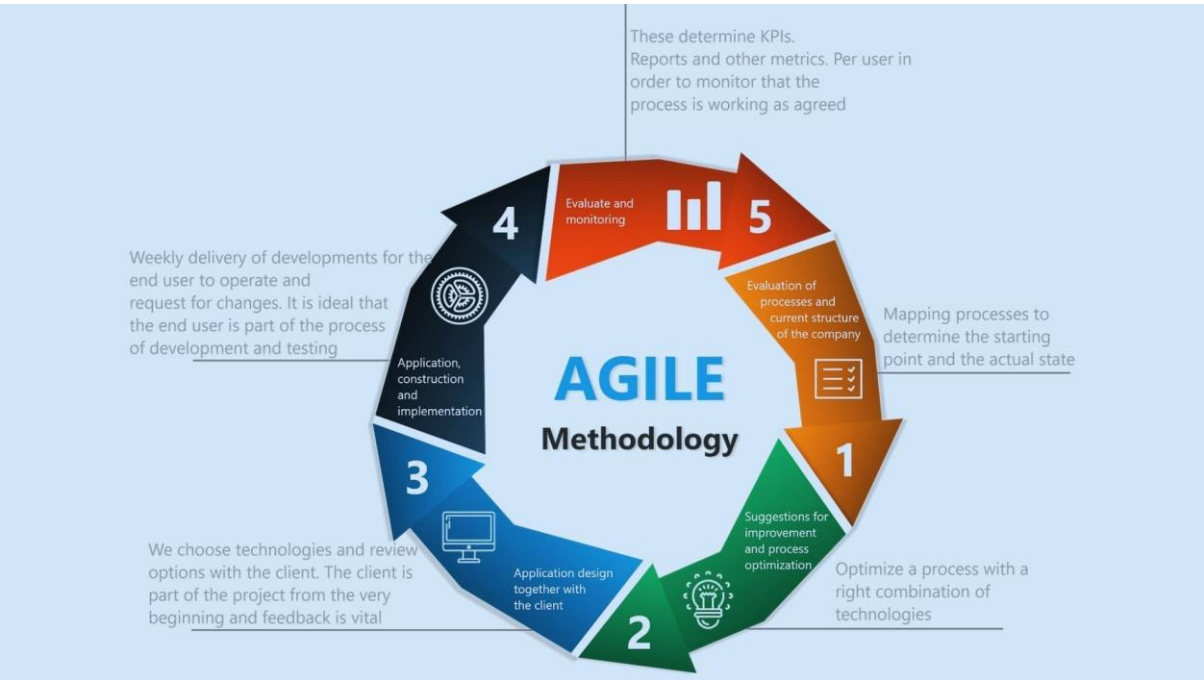


Fig 01: Methodology

3.2.2. Use Case Diagram

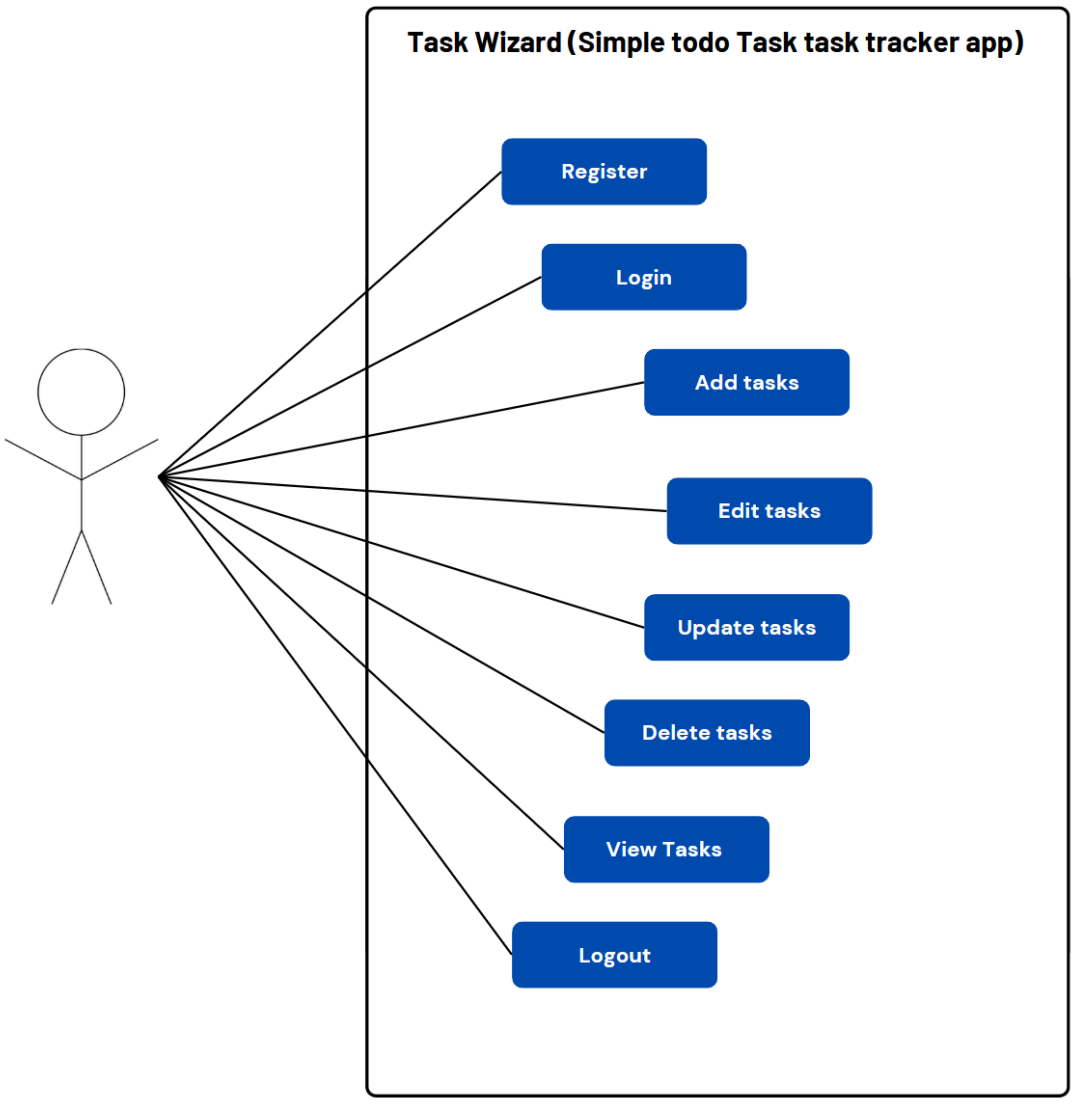


Fig 02 : Use case diagarm

3.2.3. Context Level Diagram

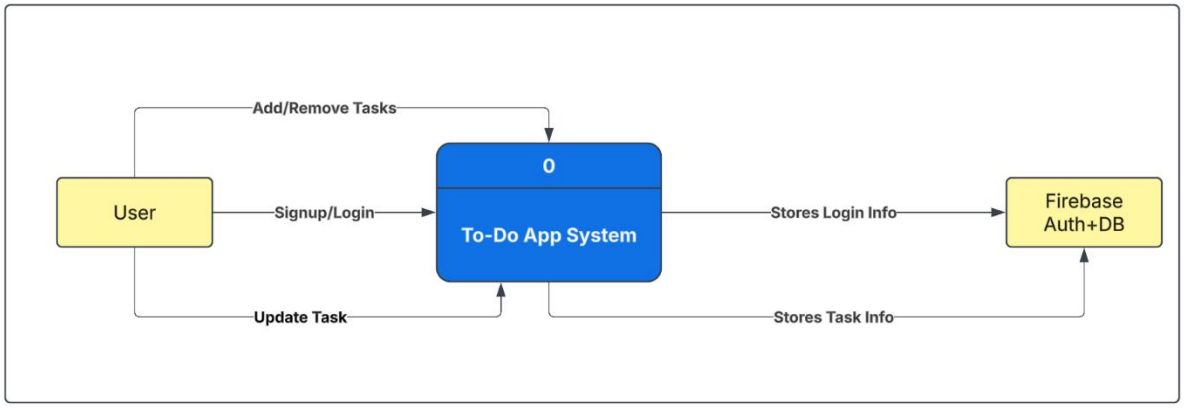


Fig 03: Context level diagram

3.2.4. Data Flow Diagram

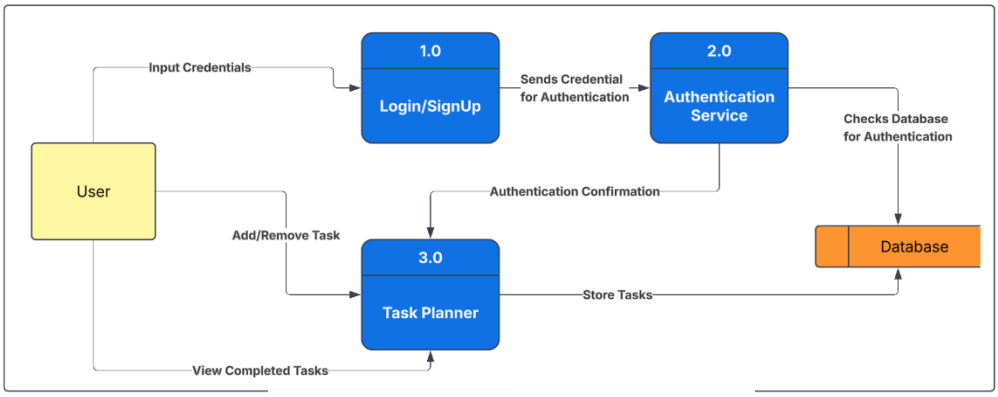


Fig 04: Dataflow diagram

3.2.5. Database Schema

users collection

userId	String
email	String
name	String
createdAt	Timestamp

tasks collection

taskId	String
userId	String
title	String
description	String
status	Boolean
dueDate	Timestamp
createdAt	Timestamp

Fig 05: Database schema

3.2.6. Flowchart

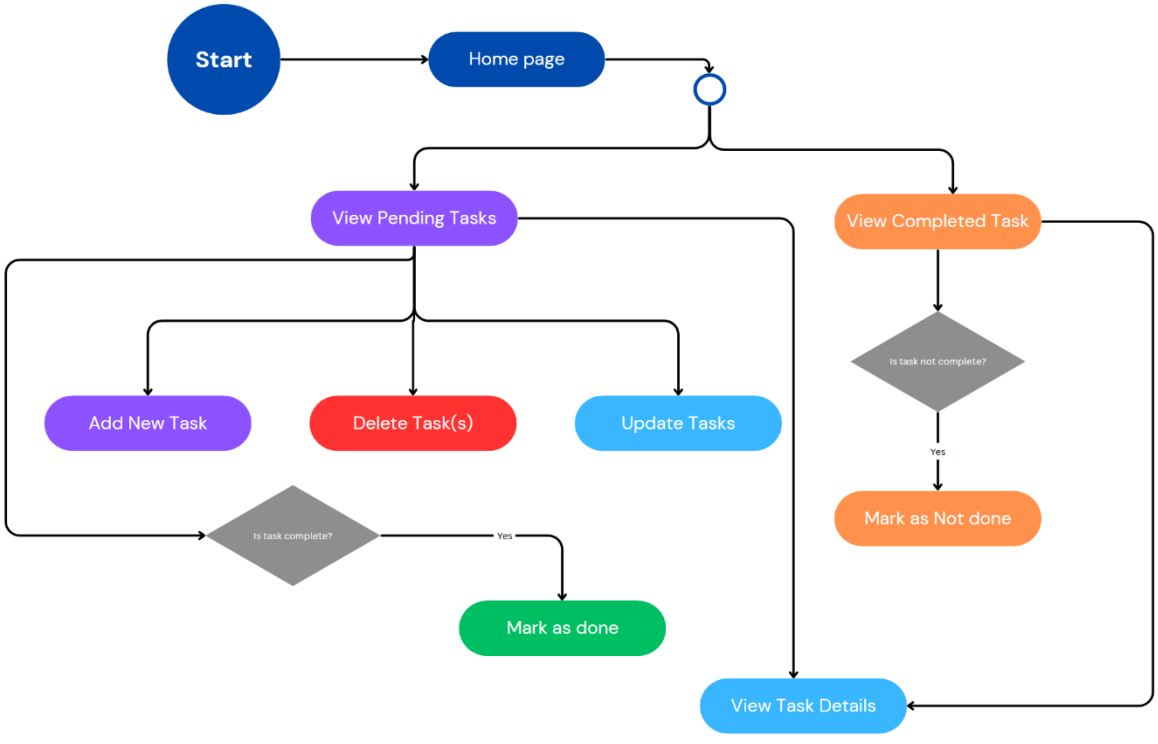


Fig 06: System flowchart

Chapter 4
IMPLEMENTATION

4.1. Interface/Front-End Design

Login_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/auth_provider.dart';
import '../providers/todo_provider.dart';
import '../widgets/custom_text_field.dart';
import '../widgets/logo.dart';
import 'signup_screen.dart';
import 'home_screen.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool _isPasswordVisible = false;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  // Validate form inputs
  bool _validateInputs() {
    if (_emailController.text.trim().isEmpty) {
      _showError('Please enter your email');
      return false;
    }
    if (!RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-
]{2,4}$').hasMatch(_emailController.text.trim())) {
      _showError('Please enter a valid email');
      return false;
    }
    if (_passwordController.text.isEmpty) {
      _showError('Please enter your password');
```

```
      return false;
    }
    return true;
  }

  // Show error message
  void _showError(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(message),
        backgroundColor: Colors.red,
      ),
    );
  }

  // Sign in user
  Future<void> _signIn() async {
    if (!_validateInputs()) return;

    final authProvider = Provider.of<AuthProvider>(context, listen: false);

    final success = await authProvider.signIn(
      email: _emailController.text.trim(),
      password: _passwordController.text,
    );

    if (success && mounted) {
      print('Login successful, navigating to HomeScreen');

      // Ensure TodoProvider is ready for the new user
      if (authProvider.userId != null) {
        final todoProvider = Provider.of<TodoProvider>(context, listen:
false);
        todoProvider.clearTodos(); // Clear any previous todos
        // Note: We don't need to call initTodos here as HomeScreen will do
that in its initState
      }

      // Navigate to home screen on successful login
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (context) => const HomeScreen()),
      );
    } else if (!success && mounted && authProvider.error != null) {
      _showError(authProvider.error!);
    }
  }

  @override
```

```
  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);

    // Check if already authenticated
    if (authProvider.isAuthenticated) {
      print('User already authenticated, navigating to HomeScreen');
      // Use Future.microtask to avoid calling Navigator during build
      Future.microtask(() {
        Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => const HomeScreen()),
        );
      });
    }

    return Scaffold(
      backgroundColor: Colors.white,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 24.0),
          child: Column(
            children: [
              const SizedBox(height: 40),
              // Logo
              const LogoWidget(),

              const Spacer(),

              // Login/Signup tabs
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  TextButton(
                    onPressed: () {
                      Navigator.of(context).pushReplacement(
                        MaterialPageRoute(builder: (context) => const
SignupScreen()),

                      );
                    },
                    child: const Text(
                      'Signup',
                      style: TextStyle(
                        fontSize: 18,
                        color: Colors.grey,
                      ),
                    ),
                  ),
                  const SizedBox(width: 30),
                  TextButton(
```

```
                    onPressed: () {
                      // Already on login page
                    },
                    child: const Text(
                      'Login',
                      style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.bold,
                        color: Colors.black,
                      ),
                    ),
                  ),
                ],
              ),

              const SizedBox(height: 40),

              // Email field
              CustomTextField(
                controller: _emailController,
                hintText: 'Email',
                keyboardType: TextInputType.emailAddress,
                prefixIcon: null,
                suffixIcon: IconButton(
                  icon: const Icon(Icons.close, color: Colors.grey),
                  onPressed: () {
                    _emailController.clear();
                  },
                ),
              ),

              const SizedBox(height: 16),

              // Password field
              CustomTextField(
                controller: _passwordController,
                hintText: 'Password',
                obscureText: !_isPasswordVisible,
                prefixIcon: null,
                suffixIcon: IconButton(
                  icon: Icon(
                    _isPasswordVisible ? Icons.visibility :
Icons.visibility_off,
                    color: const Color(0xFF8A2BE2),
                  ),
                  onPressed: () {
                    setState(() {
                      _isPasswordVisible = !_isPasswordVisible;
```

```
            });
          },
        ),
      ),

      const Spacer(),

      // Login button
      SizedBox(
        width: double.infinity,
        height: 56,
        child: ElevatedButton(
          onPressed: authProvider.isLoading ? null : _signIn,
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFF8A2BE2),
            foregroundColor: Colors.white,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(30),
            ),
          ),
          child: authProvider.isLoading
              ? const CircularProgressIndicator(color: Colors.white)
              : const Text(
                  'Login',
                  style: TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                ),
        ),
      ),

      const SizedBox(height: 20),

      // OR divider
      Row(
        children: [
          const Expanded(child: Divider(color: Colors.grey)),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            child: Text(
              'OR',
              style: TextStyle(color: Colors.grey[600]),
            ),
          ),
          const Expanded(child: Divider(color: Colors.grey)),
        ],
      ),
```

```
              const SizedBox(height: 20),

              // Google Sign In Button
              SizedBox(
                width: double.infinity,
                height: 56,
                child: ElevatedButton.icon(
                  onPressed: authProvider.isLoading
                      ? null
                      : () async {
                          final success = await
authProvider.signInWithGoogle();
                          if (success && mounted) {
                            // Ensure TodoProvider is ready for the new user
                            if (authProvider.userId != null) {
                              final todoProvider =
Provider.of<TodoProvider>(context, listen: false);
                              todoProvider.clearTodos(); // Clear any previous
todos
                            }

                            Navigator.of(context).pushReplacement(
                              MaterialPageRoute(builder: (context) => const
HomeScreen()),
                            );
                          } else if (!success && mounted && authProvider.error
!= null) {
                            _showError(authProvider.error!);
                          }
                        },
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.white,
                    foregroundColor: Colors.black87,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(30),
                      side: const BorderSide(color: Colors.grey),
                    ),
                    elevation: 1,
                  ),
                  icon: Image.asset(
                    'assets/logo.png',
                    height: 24,
                  ),
                  label: const Text(
                    'Sign in with Google',
                    style: TextStyle(
                      fontSize: 16,
```

```
                    fontWeight: FontWeight.w500,
                  ),
                ),
              ),
            ),

            const SizedBox(height: 40),
          ],
        ),
      ),
    ),
  );
  }
}
```

Signup_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/auth_provider.dart';
import '../widgets/custom_text_field.dart';
import '../widgets/logo.dart';
import 'login_screen.dart';
import 'home_screen.dart';

class SignupScreen extends StatefulWidget {
  const SignupScreen({super.key});

  @override
  State<SignupScreen> createState() => _SignupScreenState();
}

class _SignupScreenState extends State<SignupScreen> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool _isPasswordVisible = false;

  @override
  void dispose() {
    _nameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  // Validate form inputs
  bool _validateInputs() {
    if (_nameController.text.trim().isEmpty) {
```

```
      _showError('Please enter your name');
      return false;
    }
    if (_emailController.text.trim().isEmpty) {
      _showError('Please enter your email');
      return false;
    }
    if (!RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-
]{2,4}$').hasMatch(_emailController.text.trim())) {
      _showError('Please enter a valid email');
      return false;
    }
    if (_passwordController.text.isEmpty) {
      _showError('Please enter your password');
      return false;
    }
    if (_passwordController.text.length < 6) {
      _showError('Password must be at least 6 characters');
      return false;
    }
    return true;
  }

  // Show error message
  void _showError(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(message),
        backgroundColor: Colors.red,
      ),
    );
  }

  // Sign up user
  Future<void> _signUp() async {
    if (!_validateInputs()) return;

    final authProvider = Provider.of<AuthProvider>(context, listen: false);

    final success = await authProvider.signUp(
      email: _emailController.text.trim(),
      password: _passwordController.text,
      name: _nameController.text.trim(),
    );

    if (success && mounted) {
      print('Signup successful, navigating to HomeScreen');
      // Navigate to home screen on successful signup
```

```
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (context) => const HomeScreen()),
      );
    } else if (!success && mounted && authProvider.error != null) {
      _showError(authProvider.error!);
    }
  }

  @override
  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);

    // Check if already authenticated
    if (authProvider.isAuthenticated) {
      print('User already authenticated, navigating to HomeScreen');
      // Use Future.microtask to avoid calling Navigator during build
      Future.microtask(() {
        Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => const HomeScreen()),
        );
      });
    }

    return Scaffold(
      backgroundColor: Colors.white,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 24.0),
          child: Column(
            children: [
              const SizedBox(height: 40),
              // Logo
              const LogoWidget(),

              const Spacer(),

              // Login/Signup tabs
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  TextButton(
                    onPressed: () {
                      // Already on signup page
                    },
                    child: const Text(
                      'Signup',
                      style: TextStyle(
                        fontSize: 18,
```

```
                        fontWeight: FontWeight.bold,
                        color: Colors.black,
                      ),
                    ),
                  ),
                  const SizedBox(width: 30),
                  TextButton(
                    onPressed: () {
                      Navigator.of(context).pushReplacement(
                        MaterialPageRoute(builder: (context) => const
LoginScreen()),
                      );
                    },
                    child: const Text(
                      'Login',
                      style: TextStyle(
                        fontSize: 18,
                        color: Colors.grey,
                      ),
                    ),
                  ),
                ],
              ),

              const SizedBox(height: 40),

              // Name field
              CustomTextField(
                controller: _nameController,
                hintText: 'Full Name',
                keyboardType: TextInputType.name,
                prefixIcon: null,
                suffixIcon: IconButton(
                  icon: const Icon(Icons.close, color: Colors.grey),
                  onPressed: () {
                    _nameController.clear();
                  },
                ),
              ),

              const SizedBox(height: 16),

              // Email field
              CustomTextField(
                controller: _emailController,
                hintText: 'Email',
                keyboardType: TextInputType.emailAddress,
                prefixIcon: null,
```

```
                suffixIcon: IconButton(
                  icon: const Icon(Icons.close, color: Colors.grey),
                  onPressed: () {
                    _emailController.clear();
                  },
                ),
              ),

              const SizedBox(height: 16),

              // Password field
              CustomTextField(
                controller: _passwordController,
                hintText: 'Password',
                obscureText: !_isPasswordVisible,
                prefixIcon: null,
                suffixIcon: IconButton(
                  icon: Icon(
                    _isPasswordVisible ? Icons.visibility :
Icons.visibility_off,
                    color: const Color(0xFF8A2BE2),
                  ),
                  onPressed: () {
                    setState(() {
                      _isPasswordVisible = !_isPasswordVisible;
                    });
                  },
                ),
              ),

              const Spacer(),

              // Signup button
              SizedBox(
                width: double.infinity,
                height: 56,
                child: ElevatedButton(
                  onPressed: authProvider.isLoading ? null : _signUp,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: const Color(0xFF8A2BE2),
                    foregroundColor: Colors.white,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(30),
                    ),
                  ),
                  child: authProvider.isLoading
                      ? const CircularProgressIndicator(color: Colors.white)
                      : const Text(
```

```
                          'Signup',
                          style: TextStyle(
                            fontSize: 18,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                      ),
                    ),
                  ),

                  const SizedBox(height: 40),
                ],
              ),
            ),
          ),
        ),
      );
  }
}
```

Home_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter/foundation.dart';
import 'package:provider/provider.dart';
import 'package:uuid/uuid.dart';
import '../models/todo_model.dart';
import '../providers/auth_provider.dart';
import '../providers/todo_provider.dart';
import '../widgets/todo_item.dart';
import 'add_todo_screen.dart';
import 'login_screen.dart';

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  int _selectedIndex = 0;
  final Uuid _uuid = const Uuid();

  @override
  void initState() {
    super.initState();
    // Initialize todos when HomeScreen is created
    Future.microtask(() {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
```

```
      final todoProvider = Provider.of<TodoProvider>(context, listen: false);

      if (authProvider.isAuthenticated && authProvider.userId != null) {
        if (kDebugMode) {
          print('HomeScreen initState: Initializing todos for user:
${authProvider.userId}');
        }
        todoProvider.initTodos(authProvider.userId!);
      }
    });
  }

  void _showTaskDetails(BuildContext context, Todo todo) {
    showModalBottomSheet(
      context: context,
      isScrollControlled: true,
      backgroundColor: Colors.transparent,
      builder: (context) => TaskDetailsBottomSheet(
        todo: todo,
        onUpdateTask: (updatedTodo) {
          Provider.of<TodoProvider>(context, listen:
false).updateTodo(updatedTodo);
        },
      ),
    );
  }

  void _signOut() async {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final todoProvider = Provider.of<TodoProvider>(context, listen: false);

    await authProvider.signOut();
    todoProvider.clearTodos();

    if (mounted) {
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (context) => const LoginScreen()),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);
    final todoProvider = Provider.of<TodoProvider>(context);

    // Get active and completed todos
    final activeTodos = todoProvider.activeTodos;
```

```
    final completedTodos = todoProvider.completedTodos;

    return Scaffold(
      backgroundColor: Colors.grey[100],
      appBar: AppBar(
        title: Text(
          _selectedIndex == 0 ? 'Tasks to do' : 'Completed tasks',
          style: const TextStyle(
            fontWeight: FontWeight.bold,
          ),
        ),
        backgroundColor: const Color(0xFF8A2BE2),
        foregroundColor: Colors.white,
        elevation: 0,
        actions: [
          // Show user's name
          if (authProvider.userName != null)
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16),
              child: Center(
                child: Text(
                  'Hello, ${authProvider.userName}',
                  style: const TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
          // Sign out button
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: _signOut,
            tooltip: 'Sign out',
          ),
        ],
      ),
      body: todoProvider.isLoading
          ? const Center(child: CircularProgressIndicator())
          : _selectedIndex == 0
              ? _buildTodoList(context, activeTodos)
              : _buildTodoList(context, completedTodos),
      floatingActionButton: FloatingActionButton(
        onPressed: () async {
          final result = await Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => const AddTodoScreen(),
            ),
```

```
          );

          if (result != null && result is Todo) {
            // Create a new todo with a unique ID
            final newTodo = Todo(
              id: _uuid.v4(),
              title: result.title,
              description: result.description,
              createdAt: DateTime.now(),
              dueDate: result.dueDate,
            );

            await Provider.of<TodoProvider>(context, listen:
false).addTodo(newTodo);
          }
        },
        backgroundColor: const Color(0xFF8A2BE2),
        foregroundColor: Colors.white,
        child: const Icon(Icons.add),
      ),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _selectedIndex,
        onTap: (index) {
          setState(() {
            _selectedIndex = index;
          });
        },
        items: [
          BottomNavigationBarItem(
            icon: Icon(
              Icons.format_list_bulleted_rounded,
              color: _selectedIndex == 0 ? const Color(0xFF8A2BE2) :
Colors.grey,
            ),
            label: 'Tasks to do',
          ),
          BottomNavigationBarItem(
            icon: Icon(
              Icons.check_circle_outline,
              color: _selectedIndex == 1 ? const Color(0xFF8A2BE2) :
Colors.grey,
            ),
            label: 'Completed',
          ),
        ],
        selectedItemColor: const Color(0xFF8A2BE2),
      ),
    );
```

```
  }

  Widget _buildTodoList(BuildContext context, List<Todo> todoList) {
    final todoProvider = Provider.of<TodoProvider>(context, listen: false);

    return todoList.isEmpty
        ? Center(
            child: Text(
              _selectedIndex == 0
                ? 'No tasks yet. Add a new task!'
                : 'No completed tasks yet.',
              style: const TextStyle(
                fontSize: 18,
                color: Colors.grey,
              ),
            ),
          )
        : ListView.builder(
            padding: const EdgeInsets.all(16),
            itemCount: todoList.length,
            itemBuilder: (context, index) {
              final todo = todoList[index];
              return Padding(
                padding: const EdgeInsets.only(bottom: 12),
                child: GestureDetector(
                  onTap: () => _showTaskDetails(context, todo),
                  child: TodoItem(
                    todo: todo,
                    onToggle: () => todoProvider.toggleTodoCompletion(todo),
                    onDelete: () => todoProvider.deleteTodo(todo.id),
                  ),
                ),
              );
            },
          );
  }
}

class TaskDetailsBottomSheet extends StatefulWidget {
  final Todo todo;
  final Function(Todo) onUpdateTask;

  const TaskDetailsBottomSheet({
    super.key,
    required this.todo,
    required this.onUpdateTask,
  });
```

```
  @override
  State<TaskDetailsBottomSheet> createState() =>
_TaskDetailsBottomSheetState();
}

class _TaskDetailsBottomSheetState extends State<TaskDetailsBottomSheet> {
  late TextEditingController _titleController;
  late TextEditingController _descriptionController;
  late DateTime? _dueDate;

  @override
  void initState() {
    super.initState();
    _titleController = TextEditingController(text: widget.todo.title);
    _descriptionController = TextEditingController(text:
widget.todo.description);
    _dueDate = widget.todo.dueDate;
  }

  @override
  void dispose() {
    _titleController.dispose();
    _descriptionController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.only(
        top: 20,
        right: 20,
        left: 20,
        bottom: MediaQuery.of(context).viewInsets.bottom + 20,
      ),
      decoration: const BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.vertical(top: Radius.circular(20)),
      ),
      child: SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          mainAxisSize: MainAxisSize.min,
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                const Text(
```

```
                  'Edit Task',
                  style: TextStyle(
                    fontSize: 20,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                IconButton(
                  icon: const Icon(Icons.close),
                  onPressed: () => Navigator.pop(context),
                ),
              ],
            ),
            const SizedBox(height: 20),
            TextField(
              controller: _titleController,
              decoration: const InputDecoration(
                labelText: 'Task Title',
                border: OutlineInputBorder(),
                floatingLabelBehavior: FloatingLabelBehavior.always,
              ),
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _descriptionController,
              decoration: const InputDecoration(
                labelText: 'Description',
                border: OutlineInputBorder(),
                floatingLabelBehavior: FloatingLabelBehavior.always,
              ),
              maxLines: 3,
            ),
            const SizedBox(height: 16),
            Row(
              children: [
                Text(
                  'Due Date: ${_dueDate != null ?
'${_dueDate!.day}/${_dueDate!.month}/${_dueDate!.year}' : 'None'}',
                  style: const TextStyle(fontSize: 16),
                ),
                const Spacer(),
                TextButton(
                  onPressed: () async {
                    final DateTime? picked = await showDatePicker(
                      context: context,
                      initialDate: _dueDate ?? DateTime.now(),
                      firstDate: DateTime.now(),
                      lastDate: DateTime.now().add(const Duration(days: 365)),
                    );
```

```
                    if (picked != null) {
                      setState(() {
                        _dueDate = picked;
                      });
                    }
                  },
                  child: const Text('Change'),
                ),
                TextButton(
                  onPressed: () {
                    setState(() {
                      _dueDate = null;
                    });
                  },
                  child: const Text('Clear'),
                ),
              ],
            ),
            const SizedBox(height: 20),
            SizedBox(
              width: double.infinity,
              height: 50,
              child: ElevatedButton(
                onPressed: () {
                  final updatedTodo = widget.todo.copyWith(
                    title: _titleController.text,
                    description: _descriptionController.text,
                    dueDate: _dueDate,
                  );
                  widget.onUpdateTask(updatedTodo);
                  Navigator.pop(context);
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFF8A2BE2),
                  foregroundColor: Colors.white,
                ),
                child: const Text(
                  'Save Changes',
                  style: TextStyle(fontSize: 16),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Add_todo_screen.dart:

```dart
import 'package:flutter/material.dart';
import '../models/todo_model.dart';

class AddTodoScreen extends StatefulWidget {
  const AddTodoScreen({super.key});

  @override
  State<AddTodoScreen> createState() => _AddTodoScreenState();
}

class _AddTodoScreenState extends State<AddTodoScreen> {
  final _titleController = TextEditingController();
  final _descriptionController = TextEditingController();
  DateTime? _dueDate;
  final _formKey = GlobalKey<FormState>();

  @override
  void dispose() {
    _titleController.dispose();
    _descriptionController.dispose();
    super.dispose();
  }

  void _selectDate() async {
    final pickedDate = await showDatePicker(
      context: context,
      initialDate: DateTime.now(),
      firstDate: DateTime.now(),
      lastDate: DateTime.now().add(const Duration(days: 365)),
    );

    if (pickedDate != null) {
      setState(() {
        _dueDate = pickedDate;
      });
    }
  }

  void _saveTodo() {
    if (_formKey.currentState!.validate()) {
      final newTodo = Todo(
        // id is now generated in the HomeScreen
        id: 'temp-id',
        title: _titleController.text,
        description: _descriptionController.text,
        createdAt: DateTime.now(),
```

```
      dueDate: _dueDate,
    );

    Navigator.pop(context, newTodo);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
      leading: IconButton(
        icon: const Icon(Icons.close),
        onPressed: () => Navigator.pop(context),
      ),
      title: const Text('Add Task'),
      backgroundColor: Colors.white,
      foregroundColor: Colors.black,
      elevation: 0,
    ),
    body: Container(
      padding: const EdgeInsets.all(20.0),
      color: Colors.white,
      child: Form(
        key: _formKey,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFormField(
              controller: _titleController,
              decoration: InputDecoration(
                hintText: 'Adding task - form focus',
                hintStyle: TextStyle(color: Colors.grey[400]),
                enabledBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10),
                  borderSide: BorderSide(color: Colors.grey[300]!),
                ),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10),
                  borderSide: const BorderSide(color: Color(0xFF8A2BE2)),
                ),
                errorBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10),
                  borderSide: const BorderSide(color: Colors.red),
                ),
                focusedErrorBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(10),
```

```
            borderSide: const BorderSide(color: Colors.red),
          ),
          contentPadding: const EdgeInsets.all(15),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter a title';
          }
          return null;
        },
      ),
      const SizedBox(height: 16),
      TextFormField(
        controller: _descriptionController,
        decoration: InputDecoration(
          hintText: 'Details...',
          hintStyle: TextStyle(color: Colors.grey[400]),
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: BorderSide(color: Colors.grey[300]!),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: const BorderSide(color: Color(0xFF8A2BE2)),
          ),
          errorBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: const BorderSide(color: Colors.red),
          ),
          focusedErrorBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: const BorderSide(color: Colors.red),
          ),
          contentPadding: const EdgeInsets.all(15),
        ),
        maxLines: 5,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter a description';
          }
          return null;
        },
      ),
      const SizedBox(height: 20),
      // Due date selection
      Row(
        children: [
          Text(
```

```
                    'Due Date: ${_dueDate != null ?
'${_dueDate!.day}/${_dueDate!.month}/${_dueDate!.year}' : 'None'}',
                    style: const TextStyle(fontSize: 16),
                  ),
                  const Spacer(),
                  TextButton(
                    onPressed: _selectDate,
                    child: const Text('Set Due Date'),
                  ),
                  if (_dueDate != null)
                    TextButton(
                      onPressed: () {
                        setState(() {
                          _dueDate = null;
                        });
                      },
                      child: const Text('Clear'),
                    ),
                ],
              ),
              const Spacer(),
              SizedBox(
                width: double.infinity,
                height: 56,
                child: ElevatedButton(
                  onPressed: _saveTodo,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: const Color(0xFF8A2BE2),
                    foregroundColor: Colors.white,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(10),
                    ),
                    padding: const EdgeInsets.symmetric(vertical: 15),
                  ),
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: const [
                      Icon(Icons.add),
                      SizedBox(width: 8),
                      Text(
                        'Add Task',
                        style: TextStyle(
                          fontSize: 18,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ],
                  ),
```

```
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

We designed Task Wizard with a clean, modern interface focused on usability and visual appeal:

- **Color Scheme**: Used a primary purple color (#8A2BE2) for branding with white backgrounds for readability
- **Navigation**: Implemented bottom navigation for switching between active and completed tasks
- **Task Cards**: Created visually distinct cards with completion checkboxes and action buttons
- **Authentication UI**: Designed minimalist login/signup screens with the app logo and form fields
- **Responsive Design**: Ensured the UI adapts to different screen sizes with proper padding and spacing
- **Visual Feedback**: Added loading indicators and error messages for user actions

The UI follows Material Design principles with custom styling for a unique app identity.

4.2. Interface/Back-End Design

Auth_services.dart:

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/foundation.dart';
import 'package:google_sign_in/google_sign_in.dart';

class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final GoogleSignIn _googleSignIn = GoogleSignIn();

  // Get current user
  User? get currentUser => _auth.currentUser;

  // Get user stream
  Stream<User?> get userStream => _auth.authStateChanges();

  // Sign up with email and password
  Future<UserCredential> signUp({
    required String email,
    required String password,
    required String name,
  }) async {
    try {
      if (kDebugMode) {
        print('Attempting to create user with email: $email');
      }
```

```
      // Create user with email and password
      final userCredential = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );

      if (kDebugMode) {
        print('User created successfully with ID:
${userCredential.user?.uid}');
      }

      // Update display name
      await userCredential.user?.updateDisplayName(name);

      if (kDebugMode) {
        print('Display name updated to: $name');
        print('Current user after signup: ${_auth.currentUser?.uid}');
        print('Is user email verified: ${_auth.currentUser?.emailVerified}');
      }

      return userCredential;
    } catch (e) {
      if (kDebugMode) {
        print('Sign up error: $e');
      }
      rethrow;
    }
  }

  // Sign in with email and password
  Future<UserCredential> signIn({
    required String email,
    required String password,
  }) async {
    try {
      if (kDebugMode) {
        print('Attempting to sign in with email: $email');
      }

      final userCredential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );

      if (kDebugMode) {
        print('Sign in successful for user ID: ${userCredential.user?.uid}');
        print('User display name: ${userCredential.user?.displayName}');
```

```
        print('User email verified: ${userCredential.user?.emailVerified}');
      }

      return userCredential;
    } catch (e) {
      if (kDebugMode) {
        print('Sign in error: $e');
        print('Detailed error info: ${e is FirebaseAuthException ? 'Code:
${e.code}, Message: ${e.message}' : e.toString()}');
      }
      rethrow;
    }
  }

  // Google Sign In
  Future<UserCredential?> signInWithGoogle() async {
    try {
      // Trigger the Google Authentication flow
      final GoogleSignInAccount? googleUser = await _googleSignIn.signIn();

      if (googleUser == null) {
        // User canceled the sign-in flow
        if (kDebugMode) {
          print('Google Sign-In canceled by user');
        }
        return null;
      }

      // Obtain the authentication details from the Google Sign In
      final GoogleSignInAuthentication googleAuth = await
googleUser.authentication;

      // Create a credential from the Google authentication token
      final AuthCredential credential = GoogleAuthProvider.credential(
        accessToken: googleAuth.accessToken,
        idToken: googleAuth.idToken,
      );

      // Sign in to Firebase with the Google credential
      final UserCredential userCredential = await
_auth.signInWithCredential(credential);

      if (kDebugMode) {
        print('Google Sign-In successful. User:
${userCredential.user?.displayName}');
        print('Google Sign-In user ID: ${userCredential.user?.uid}');
        print('Google Sign-In user email: ${userCredential.user?.email}');
      }
```

```
      return userCredential;
    } catch (e) {
      if (kDebugMode) {
        print('Google Sign-In error: $e');
      }
      return null;
    }
  }

  // Sign out
  Future<void> signOut() async {
    if (kDebugMode) {
      print('Signing out user: ${_auth.currentUser?.uid}');
    }

    // Sign out from Google if signed in with Google
    try {
      await _googleSignIn.signOut();
    } catch (e) {
      if (kDebugMode) {
        print('Error signing out from Google: $e');
      }
    }

    // Sign out from Firebase
    await _auth.signOut();

    if (kDebugMode) {
      print('Sign out complete. Current user is now: ${_auth.currentUser?.uid
?? 'null'}');
    }
  }

  // Get user name
  String? getUserName() {
    return _auth.currentUser?.displayName;
  }

  // Get user email
  String? getUserEmail() {
    return _auth.currentUser?.email;
  }

  // Get user ID
  String? getUserId() {
    return _auth.currentUser?.uid;
  }
```

```
  // Reload user data from Firebase
  Future<void> reloadUser() async {
    try {
      if (_auth.currentUser != null) {
        if (kDebugMode) {
          print('Reloading user data for user: ${_auth.currentUser?.uid}');
        }
        await _auth.currentUser?.reload();
        if (kDebugMode) {
          print('User data reloaded. Display name:
${_auth.currentUser?.displayName}');
        }
      }
    } catch (e) {
      if (kDebugMode) {
        print('Error reloading user data: $e');
      }
    }
  }
}
```

Firestore_services.dart:

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/foundation.dart';
import '../models/todo_model.dart';

class FirestoreService {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  // Collection reference for todos
  CollectionReference _todosCollection(String userId) {
    return _firestore.collection('users').doc(userId).collection('todos');
  }

  // Get user's todos stream
  Stream<List<Todo>> getTodos(String userId) {
    if (kDebugMode) {
      print('Getting todos for user: $userId');
    }

    return _todosCollection(userId)
        .orderBy('createdAt', descending: true)
        .snapshots()
        .map((snapshot) {
      if (kDebugMode) {
        print('Received ${snapshot.docs.length} todos from Firestore');
      }
```

```
      return snapshot.docs.map((doc) {
        final data = doc.data() as Map<String, dynamic>;
        return Todo.fromJson(data);
      }).toList();
    });
  }

  // Add a new todo
  Future<void> addTodo(String userId, Todo todo) async {
    if (kDebugMode) {
      print('Adding todo to Firestore - User: $userId, Todo ID: ${todo.id},
Title: ${todo.title}');
    }

    await _todosCollection(userId).doc(todo.id).set(todo.toJson());

    if (kDebugMode) {
      print('Todo added successfully to Firestore');
    }
  }

  // Update a todo
  Future<void> updateTodo(String userId, Todo todo) async {
    if (kDebugMode) {
      print('Updating todo in Firestore - User: $userId, Todo ID:
${todo.id}');
    }

    await _todosCollection(userId).doc(todo.id).update(todo.toJson());

    if (kDebugMode) {
      print('Todo updated successfully in Firestore');
    }
  }

  // Delete a todo
  Future<void> deleteTodo(String userId, String todoId) async {
    if (kDebugMode) {
      print('Deleting todo from Firestore - User: $userId, Todo ID: $todoId');
    }

    await _todosCollection(userId).doc(todoId).delete();

    if (kDebugMode) {
      print('Todo deleted successfully from Firestore');
    }
  }
```

```
  // Toggle todo completion status
  Future<void> toggleTodoCompletion(String userId, Todo todo) async {
    if (kDebugMode) {
      print('Toggling todo completion in Firestore - User: $userId, Todo ID:
${todo.id}, Current state: ${todo.isCompleted}');
    }

    await _todosCollection(userId).doc(todo.id).update({
      'isCompleted': !todo.isCompleted,
    });

    if (kDebugMode) {
      print('Todo completion toggled successfully in Firestore');
    }
  }
}
```

The app's back-end is built on Firebase with a clean architecture pattern:

- **Authentication System**: Implemented using Firebase Auth with both email/password and Google Sign-In
- **Data Storage**: Used Cloud Firestore for storing user and task data in a NoSQL structure
- **State Management**: Implemented Provider pattern for app-wide state management
- **CRUD Operations**:
  - Create: Adding new tasks with title, description, and due date
  - Read: Real-time synchronization of tasks from Firestore
  - Update: Editing task details and toggling completion status
  - Delete: Removing tasks from the database
- **User-Specific Data**: Implemented security rules to ensure users can only access their own tasks
- **Offline Support**: Configured Firestore for offline persistence

4.3. Modules/Features

Todo_provider.dart:

```
import 'dart:async';
import 'package:flutter/foundation.dart';
import '../models/todo_model.dart';
import '../services/firestore_service.dart';

class TodoProvider with ChangeNotifier {
  final FirestoreService _firestoreService = FirestoreService();
  List<Todo> _todos = [];
  bool _isLoading = false;
  String? _error;
  StreamSubscription? _todosSubscription;
  String? _userId;

  // Getters
  List<Todo> get todos => _todos;
  List<Todo> get activeTodos => _todos.where((todo) =>
!todo.isCompleted).toList();
  List<Todo> get completedTodos => _todos.where((todo) =>
todo.isCompleted).toList();
  bool get isLoading => _isLoading;
  String? get error => _error;

  // Initialize todos for a user
  void initTodos(String userId) {
    if (kDebugMode) {
      print('TodoProvider: Initializing todos for user: $userId');
    }

    // Cancel any existing subscription
    if (_todosSubscription != null) {
      if (kDebugMode) {
        print('TodoProvider: Cancelling previous todos subscription');
      }
      _todosSubscription!.cancel();
      _todosSubscription = null;
    }

    // Clear existing todos before loading new ones
    _todos = [];
    _userId = userId;
    _isLoading = true;
    _error = null;
    notifyListeners();

    if (kDebugMode) {
      print('TodoProvider: Setting up Firestore stream for todos');
```

```
    }

    try {
      _todosSubscription = _firestoreService.getTodos(userId).listen(
        (todos) {
          if (kDebugMode) {
            print('TodoProvider: Received ${todos.length} todos from Firestore
stream');
            if (todos.isNotEmpty) {
              print('TodoProvider: Sample todo - ID: ${todos[0].id}, Title:
${todos[0].title}');
            }
          }

          _todos = todos;
          _isLoading = false;
          notifyListeners();
        },
        onError: (e) {
          if (kDebugMode) {
            print('TodoProvider: Error fetching todos: $e');
          }
          _error = 'Error fetching your tasks';
          _isLoading = false;
          notifyListeners();
        }
      );
    } catch (e) {
      if (kDebugMode) {
        print('TodoProvider: Error setting up todos stream: $e');
      }
      _error = 'Error connecting to database';
      _isLoading = false;
      notifyListeners();
    }
  }

  // Clear todos when user signs out
  void clearTodos() {
    if (kDebugMode) {
      print('TodoProvider: Clearing todos for user: $_userId');
    }

    if (_todosSubscription != null) {
      _todosSubscription!.cancel();
      _todosSubscription = null;
    }
```

```
    _todos = [];
    _userId = null;
    notifyListeners();
  }

  // Add a todo
  Future<void> addTodo(Todo todo) async {
    if (_userId == null) {
      if (kDebugMode) {
        print('TodoProvider: Cannot add todo - no user ID');
      }
      return;
    }

    if (kDebugMode) {
      print('TodoProvider: Adding todo - Title: ${todo.title}, ID:
${todo.id}');
    }

    _isLoading = true;
    notifyListeners();

    try {
      await _firestoreService.addTodo(_userId!, todo);
      if (kDebugMode) {
        print('TodoProvider: Todo added successfully');
      }
      _isLoading = false;
      notifyListeners();
    } catch (e) {
      if (kDebugMode) {
        print('TodoProvider: Error adding todo: $e');
      }
      _error = 'Failed to add task';
      _isLoading = false;
      notifyListeners();
    }
  }

  // Update a todo
  Future<void> updateTodo(Todo todo) async {
    if (_userId == null) {
      if (kDebugMode) {
        print('TodoProvider: Cannot update todo - no user ID');
      }
      return;
    }
```

```
    if (kDebugMode) {
      print('TodoProvider: Updating todo - ID: ${todo.id}, Title:
${todo.title}');
    }

    _isLoading = true;
    notifyListeners();

    try {
      await _firestoreService.updateTodo(_userId!, todo);
      if (kDebugMode) {
        print('TodoProvider: Todo updated successfully');
      }
      _isLoading = false;
      notifyListeners();
    } catch (e) {
      if (kDebugMode) {
        print('TodoProvider: Error updating todo: $e');
      }
      _error = 'Failed to update task';
      _isLoading = false;
      notifyListeners();
    }
  }

  // Delete a todo
  Future<void> deleteTodo(String todoId) async {
    if (_userId == null) {
      if (kDebugMode) {
        print('TodoProvider: Cannot delete todo - no user ID');
      }
      return;
    }

    if (kDebugMode) {
      print('TodoProvider: Deleting todo - ID: $todoId');
    }

    _isLoading = true;
    notifyListeners();

    try {
      await _firestoreService.deleteTodo(_userId!, todoId);
      if (kDebugMode) {
        print('TodoProvider: Todo deleted successfully');
      }
      _isLoading = false;
      notifyListeners();
```

```
    } catch (e) {
      if (kDebugMode) {
        print('TodoProvider: Error deleting todo: $e');
      }
      _error = 'Failed to delete task';
      _isLoading = false;
      notifyListeners();
    }
  }

  // Toggle todo completion
  Future<void> toggleTodoCompletion(Todo todo) async {
    if (_userId == null) {
      if (kDebugMode) {
        print('TodoProvider: Cannot toggle todo completion - no user ID');
      }
      return;
    }

    if (kDebugMode) {
      print('TodoProvider: Toggling todo completion - ID: ${todo.id}, Current
state: ${todo.isCompleted}');
    }

    try {
      await _firestoreService.toggleTodoCompletion(_userId!, todo);
      if (kDebugMode) {
        print('TodoProvider: Todo completion toggled successfully');
      }
    } catch (e) {
      if (kDebugMode) {
        print('TodoProvider: Error toggling todo completion: $e');
      }
      _error = 'Failed to update task status';
      notifyListeners();
    }
  }

  // Clear error
  void clearError() {
    _error = null;
    notifyListeners();
  }

  @override
  void dispose() {
    if (_todosSubscription != null) {
      _todosSubscription!.cancel();
```

```
    }
    super.dispose();
  }
}
```

Todo_model.dart:

```
class Todo {
  final String id;
  final String title;
  final String description;
  final bool isCompleted;
  final DateTime createdAt;
  final DateTime? dueDate;

  Todo({
    required this.id,
    required this.title,
    required this.description,
    this.isCompleted = false,
    required this.createdAt,
    this.dueDate,
  });

  Todo copyWith({
    String? id,
    String? title,
    String? description,
    bool? isCompleted,
    DateTime? createdAt,
    DateTime? dueDate,
  }) {
    return Todo(
      id: id ?? this.id,
      title: title ?? this.title,
      description: description ?? this.description,
      isCompleted: isCompleted ?? this.isCompleted,
      createdAt: createdAt ?? this.createdAt,
      dueDate: dueDate ?? this.dueDate,
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'title': title,
      'description': description,
      'isCompleted': isCompleted,
```

```
      'createdAt': createdAt.toIso8601String(),
      'dueDate': dueDate?.toIso8601String(),
    };
  }

  factory Todo.fromJson(Map<String, dynamic> json) {
    return Todo(
      id: json['id'],
      title: json['title'],
      description: json['description'],
      isCompleted: json['isCompleted'],
      createdAt: DateTime.parse(json['createdAt']),
      dueDate: json['dueDate'] != null ? DateTime.parse(json['dueDate']) :
null,
    );
  }
}
```

Custom_text_fields.dart:

```
import 'package:flutter/material.dart';

class CustomTextField extends StatelessWidget {
  final TextEditingController controller;
  final String hintText;
  final bool obscureText;
  final TextInputType keyboardType;
  final Widget? prefixIcon;
  final Widget? suffixIcon;

  const CustomTextField({
    super.key,
    required this.controller,
    required this.hintText,
    this.obscureText = false,
    this.keyboardType = TextInputType.text,
    this.prefixIcon,
    this.suffixIcon,
  });

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 56,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(30),
        color: Colors.grey[100],
      ),
```

```
      child: TextField(
        controller: controller,
        obscureText: obscureText,
        keyboardType: keyboardType,
        style: const TextStyle(
          fontSize: 16,
          color: Colors.black,
        ),
        decoration: InputDecoration(
          hintText: hintText,
          hintStyle: TextStyle(
            color: Colors.grey[500],
            fontSize: 16,
          ),
          contentPadding: const EdgeInsets.symmetric(horizontal: 20),
          prefixIcon: prefixIcon,
          suffixIcon: suffixIcon,
          border: InputBorder.none,
          focusedBorder: InputBorder.none,
          enabledBorder: InputBorder.none,
          errorBorder: InputBorder.none,
          disabledBorder: InputBorder.none,
        ),
      ),
    );
  }
}
```

Todo_item.dart:

```
import 'package:flutter/material.dart';
import '../models/todo_model.dart';
import 'package:intl/intl.dart';

class TodoItem extends StatefulWidget {
  final Todo todo;
  final VoidCallback onToggle;
  final VoidCallback onDelete;

  const TodoItem({
    super.key,
    required this.todo,
    required this.onToggle,
    required this.onDelete,
  });

  @override
  State<TodoItem> createState() => _TodoItemState();
}
```

```dart
class _TodoItemState extends State<TodoItem> {
  bool _isHovering = false;

  @override
  Widget build(BuildContext context) {
    return MouseRegion(
      onEnter: (_) => setState(() => _isHovering = true),
      onExit: (_) => setState(() => _isHovering = false),
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(16),
          boxShadow: [
            BoxShadow(
              color: Colors.grey.withAlpha(25),
              spreadRadius: 1,
              blurRadius: 2,
              offset: const Offset(0, 1),
            ),
          ],
        ),
        child: Padding(
          padding: const EdgeInsets.all(16),
          child: Row(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              // Checkbox
              GestureDetector(
                onTap: widget.onToggle,
                child: Container(
                  height: 24,
                  width: 24,
                  decoration: BoxDecoration(
                    shape: BoxShape.circle,
                    color: widget.todo.isCompleted
                        ? const Color(0xFF8A2BE2)
                        : Colors.transparent,
                    border: Border.all(
                      color: widget.todo.isCompleted
                          ? const Color(0xFF8A2BE2)
                          : Colors.grey,
                      width: 2,
                    ),
                  ),
                  child: widget.todo.isCompleted
                      ? const Icon(
                          Icons.check,
```

```
                        size: 16,
                        color: Colors.white,
                      )
                    : null,
              ),
            ),
            const SizedBox(width: 16),

            // Content
            Expanded(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    widget.todo.title,
                    style: TextStyle(
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                      decoration: widget.todo.isCompleted
                          ? TextDecoration.lineThrough
                          : null,
                      color: widget.todo.isCompleted ? Colors.grey :
Colors.black,
                    ),
                  ),
                  const SizedBox(height: 8),
                  Text(
                    widget.todo.description,
                    style: TextStyle(
                      fontSize: 14,
                      color: widget.todo.isCompleted ? Colors.grey :
Colors.black87,
                      decoration: widget.todo.isCompleted
                          ? TextDecoration.lineThrough
                          : null,
                    ),
                  ),
                  if (widget.todo.dueDate != null) ...[
                    const SizedBox(height: 8),
                    Row(
                      children: [
                        const Icon(
                          Icons.calendar_today,
                          size: 14,
                          color: Colors.grey,
                        ),
                        const SizedBox(width: 4),
                        Text(
```

```
                            DateFormat('MMM d,
yyyy').format(widget.todo.dueDate!),
                            style: const TextStyle(
                              fontSize: 12,
                              color: Colors.grey,
                            ),
                          ),
                        ],
                      ),
                    ],
                  ],
                ),
              ),

              // Edit & Delete buttons
              Row(
                mainAxisSize: MainAxisSize.min,
                children: [
                  if (_isHovering || MediaQuery.of(context).size.width > 600)
                    IconButton(
                      icon: const Icon(
                        Icons.edit,
                        color: Color(0xFF8A2BE2),
                      ),
                      onPressed: () {
                        // This GestureDetector in HomeScreen will handle the
tap
                      },
                      tooltip: 'Edit Task',
                    ),
                  IconButton(
                    onPressed: widget.onDelete,
                    icon: const Icon(
                      Icons.delete_outline,
                      color: Colors.red,
                    ),
                    tooltip: 'Delete Task',
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

The Task Wizard app includes the following modules:

1. **Authentication Module**
    - Email/password authentication
    - Google Sign-In integration
    - User profile management

2. **Task Management Module**
    - Task creation with title, description, and due date
    - Task listing with filtering (active/completed)
    - Task editing and status updates
    - Task deletion
3. **UI Components Module**
    - Custom text fields
    - Task item cards
    - Logo and branding elements
    - Modal bottom sheets for task details
4. **Data Synchronization Module**
    - Real-time data updates using Firestore streams
    - User-specific data segregation
    - Error handling and retry mechanisms
5. **Navigation System**
    - Screen transitions
    - Tab-based navigation
    - Authentication state-based routing

Each module was developed with separation of concerns in mind, making the codebase maintainable and extensible for future features.

Chapter 5
USER MANUAL

5.1. System Requirement

5.1.1. Hardware Requirement

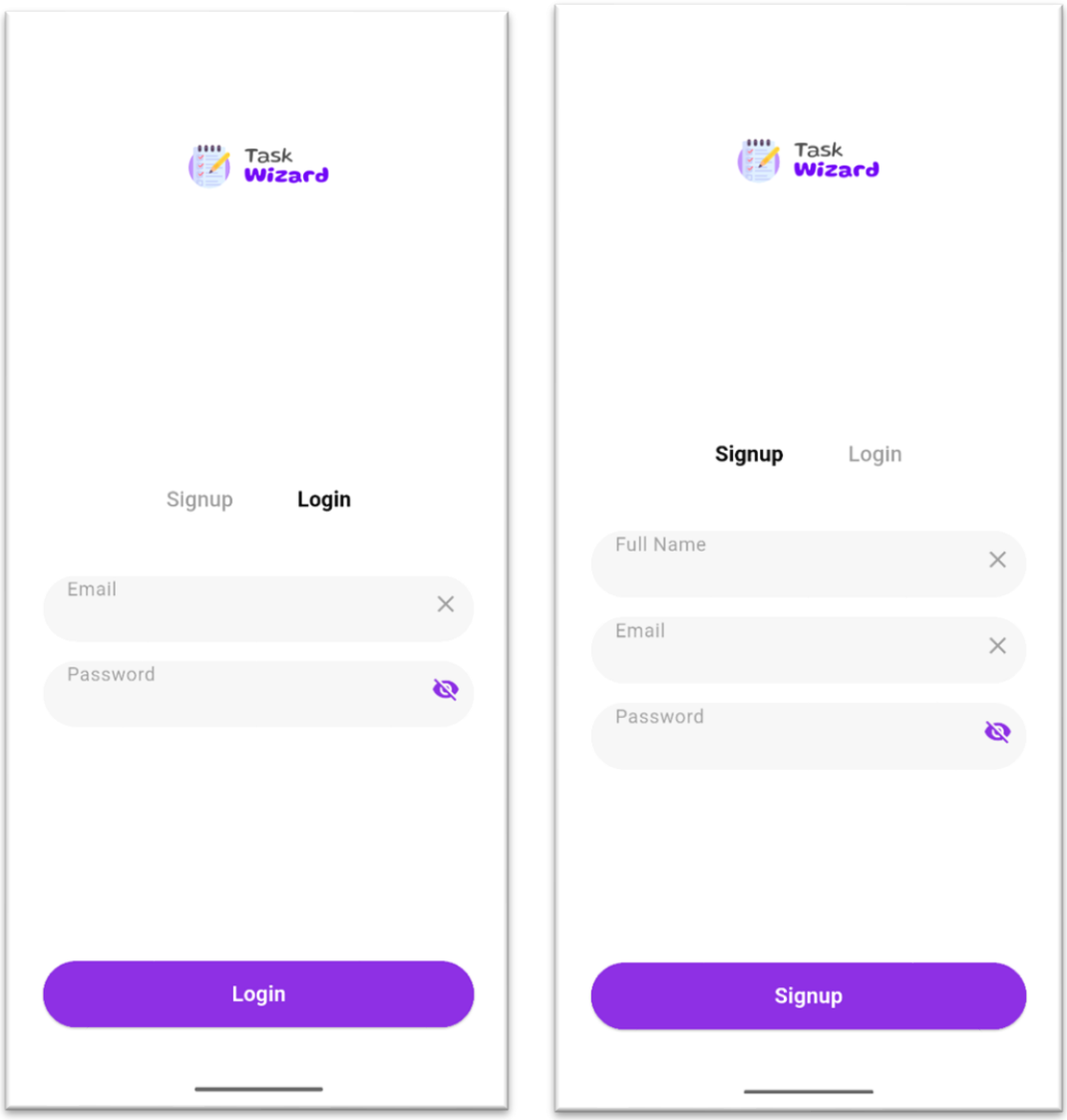
Smartphone: Android device with Android 5.0 (Lollipop) or higher
Memory: Minimum 4 GB RAM recommended for smooth performance
Storage: At least 50 MB of free space for app installation and task data
Internet: Stable internet connection (Wi-Fi or mobile data) for syncing tasks with the cloud

5.1.2. Software Requirement

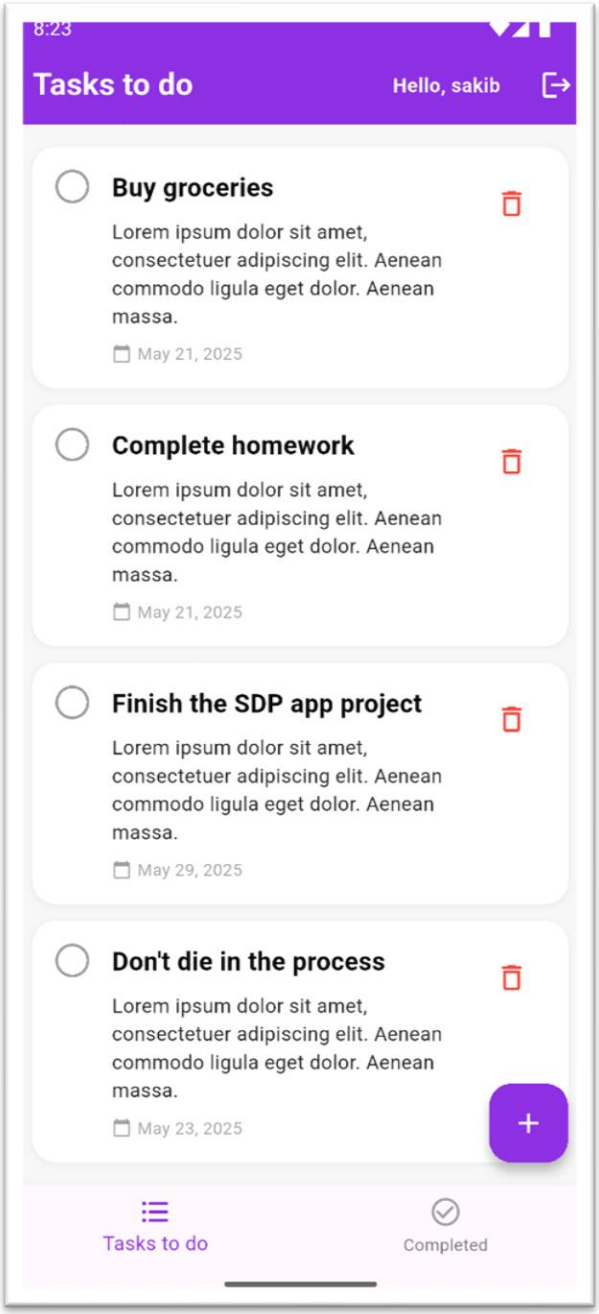
- Operating System: Android 5.0 (Lollipop) or newer
- Google Play Services: Must be enabled for Firebase support
- Internet access (for data sync)
- Notification permission (optional, for future reminder updates)

5.2. User Interfaces

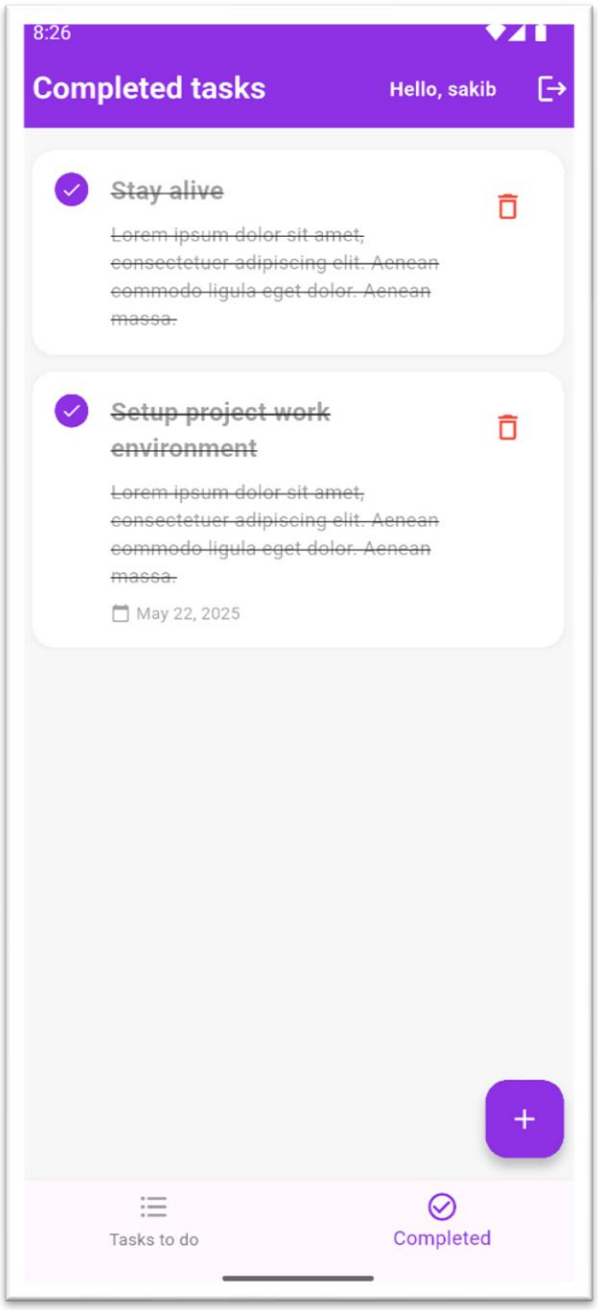
5.2.1. Panel A: Login & Signup Panel



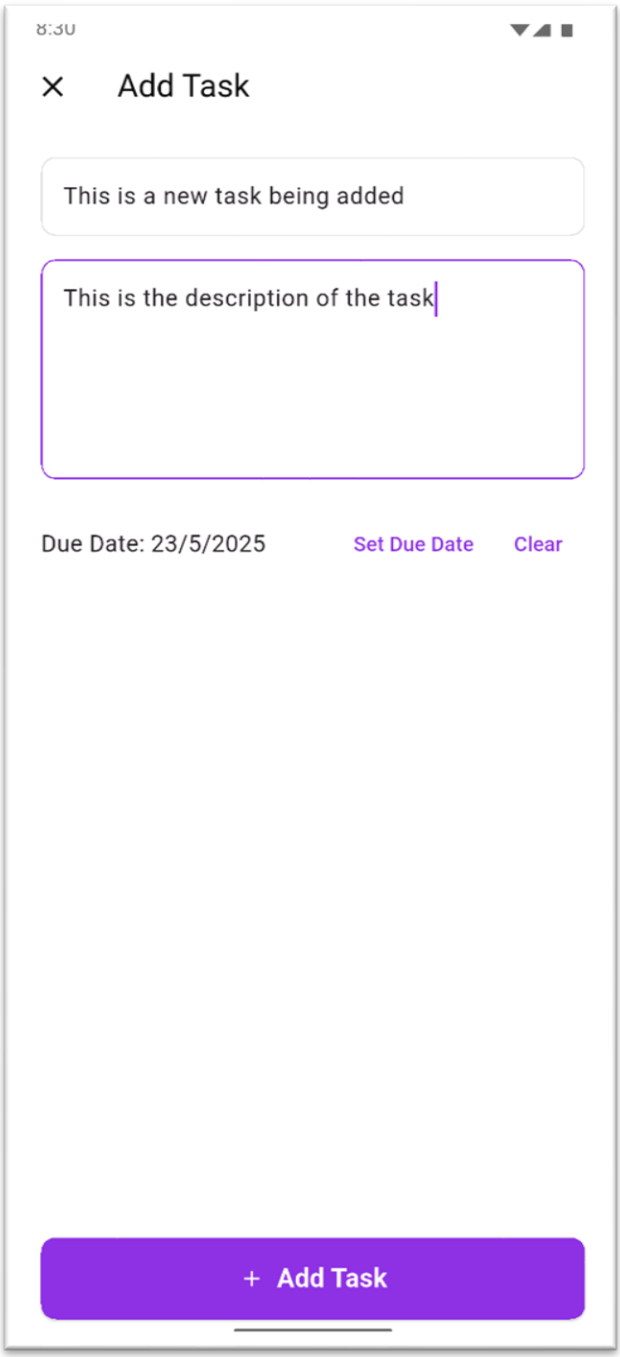
5.2.2. Panel B: Home page/Pending Tasks

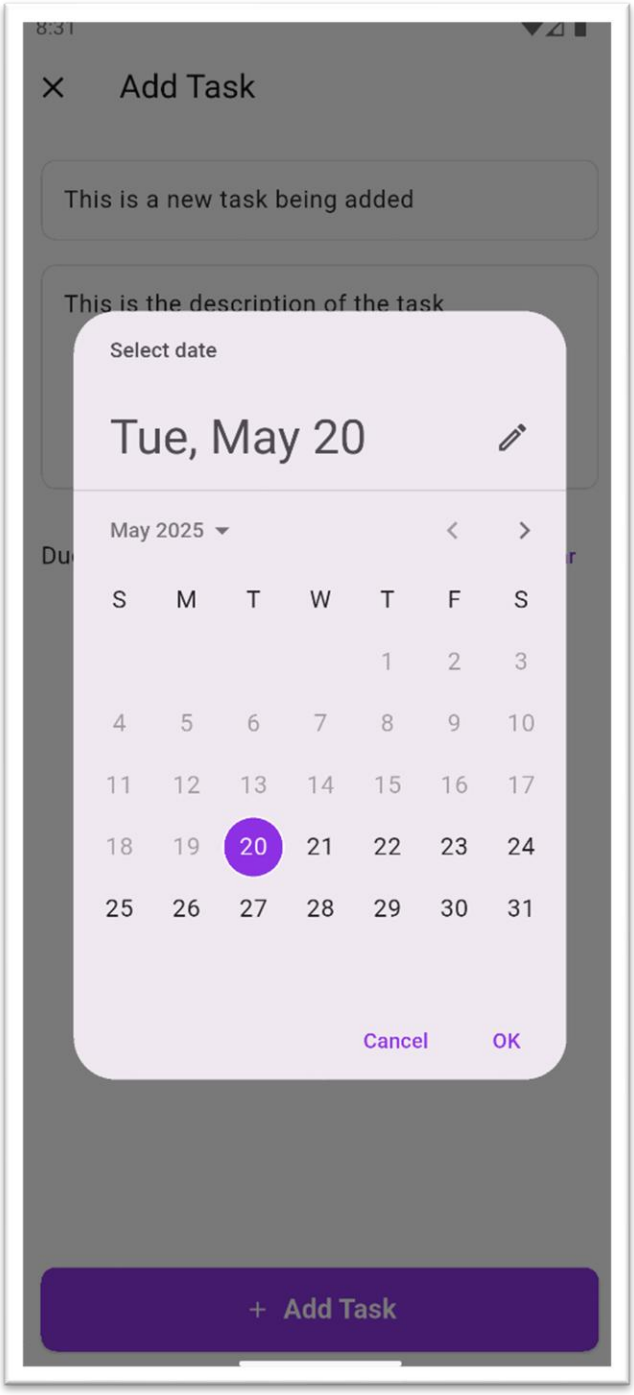


5.2.3. Pending Task

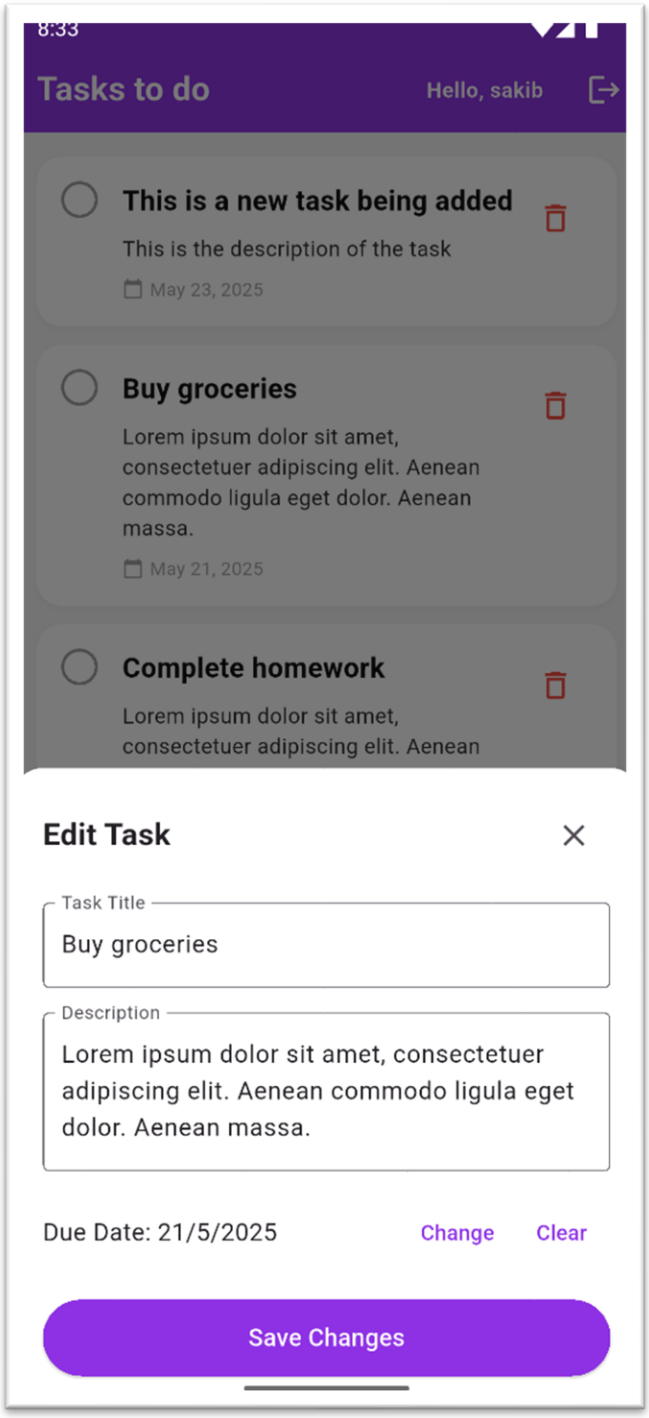


5.2.4. Adding New Task

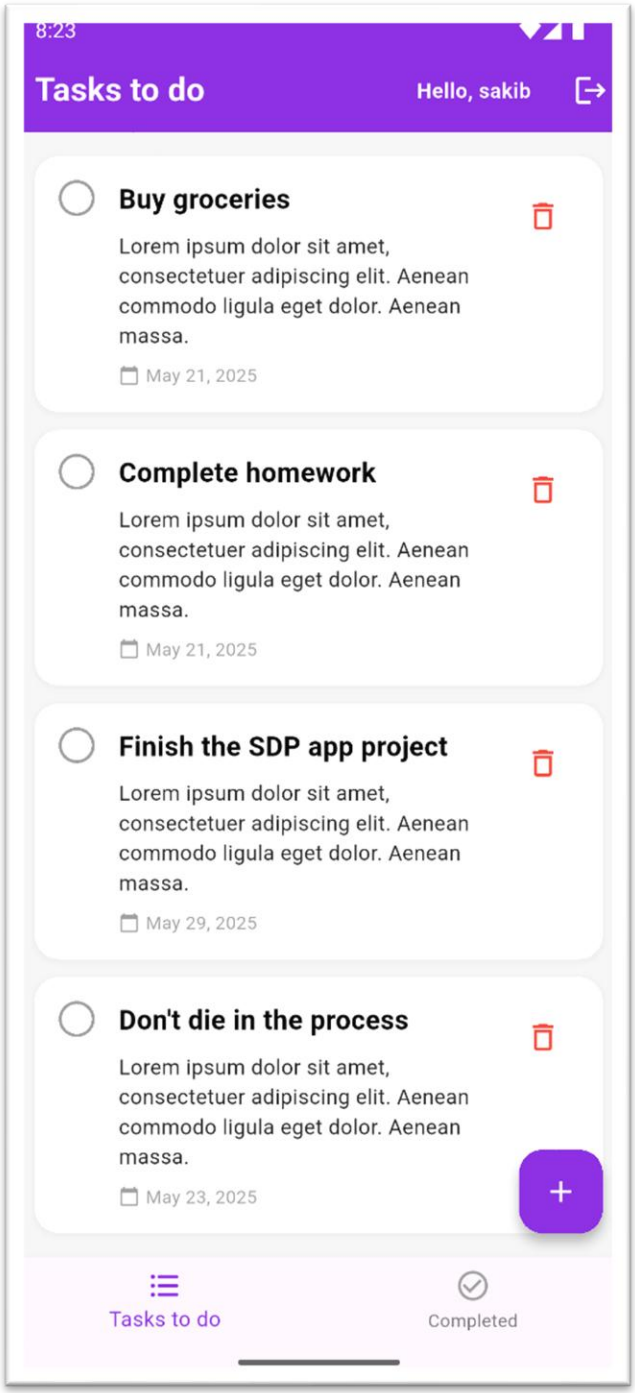




5.2.5. Editing Task



5.2.6. Deleting Task



Chapter 6
CONCLUSION

6.1. Conclusion

This project aimed to build a simple yet secure to-do list application using Flutter and Firebase. From setting up authentication to implementing task CRUD features and syncing data via Firestore, each component was carefully designed to offer a smooth user experience. The use of Flutter allowed fast UI development, and Firebase provided powerful backend support without the need for a custom server. The app fulfills the primary goal — to let users manage their tasks securely and efficiently from any device.

6.2. Limitations

- The current app supports only Android platforms.
- There are no push notifications or reminders for tasks.
- Collaboration features (like sharing tasks with others) are not included.
- Requires an active internet connection to fully function (except Firestore's limited offline mode).

6.3. Future Works

- Add push notifications to remind users of upcoming or overdue tasks.
- Build a collaboration system for shared task lists.
- Add support for iOS and web versions.
- Implement dark mode, theming, and user customization.
- Integrate calendar view, priority sorting, and voice input for tasks.

References

1. "Firebase Documentation." Firebase, Google, Accessed 20 May 2025, https://firebase.google.com/docs.
2. "Flutter Documentation." Flutter, Accessed 20 May 2025, https://docs.flutter.dev/.
3. "Material Design Guidelines." Material Design, Google, Accessed 20 May 2025, https://m3.material.io/.
4. Stonehem, Bill. *Flutter for Beginners: An Introductory Guide to Building Cross-Platform Mobile Applications with Flutter and Dart 2*. Packt Publishing, 2019.
5. Napoli, Alessandro. "Provider Pattern in Flutter." Medium, Flutter Community, 20 Jan. 2021, https://medium.com/flutter-community/provider-pattern-in-flutter-3678fce9800d.
6. "Google Sign-In for Flutter." Google Developers, Accessed 20 May 2025, https://developers.google.com/identity/sign-in/flutter.
7. "Cloud Firestore Documentation." Firebase, Google, Accessed 20 May 2025, https://firebase.google.com/docs/firestore.
8. Windmill, Eric. "State Management with Provider." Flutter, 15 Mar. 2022, https://docs.flutter.dev/development/data-and-backend/state-mgmt/simple.
9. "Flutter Packages." Pub.dev, Accessed 20 May 2025, https://pub.dev/.
10. "Task Management UI Design Inspiration." Dribbble, Accessed 20 May 2025, https://dribbble.com/tags/task_management.