A logo of a university

Description automatically generated **To-Do List Application Using Flutter and Hive**

**Project Report cum Code Logic**

**Submitted By : Piyush(22BCA115), Sahil(22BCA030), Gaurav(22BCA114)**

**Submitted To : Sumit Kumar Sir**

**A screen shot of a clipboard and a pencil

Description automatically generatedA screenshot of a phone

Description automatically generatedA screenshot of a social media post

Description automatically generated**

**Introduction**

**In today's fast-paced world, staying organized and on top of tasks is more crucial than ever. To help with this, we have developed a To-Do List Application using Flutter for the frontend and Hive for local data storage. The goal of this application is to provide a simple yet efficient tool for managing daily tasks, allowing users to add, update, delete, and filter tasks with ease.**

**Overview:**

This app is designed to help users manage their daily tasks. The app allows users to:

* Add new tasks
* Mark tasks as completed or delete them
* Filter tasks by their status (All, Completed, Pending)
* View a report of tasks

The data is stored locally using **Hive**, a lightweight database for Flutter.

**1. Home Screen: Display a List of Tasks with a Checkbox for Completion  
A screenshot of a social media post

Description automatically generated**

**Logic:**  
The Home Screen displays a list of tasks. Each task has a checkbox to mark it as completed. If a task is completed, it gets visually updated (struck-through or colored differently).

**Code Explanation:**import 'package:flutter/material.dart';

import 'package:hive/hive.dart';

import '../models/task.dart';

class HomeScreen extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('To-Do List')),

body: ValueListenableBuilder(

valueListenable: Hive.box<Task>('tasksBox').listenable(),

builder: (context, Box<Task> box, \_) {

if (box.isEmpty) return Center(child: Text('No tasks yet.'));

var tasks = box.values.toList();

return ListView.builder(

itemCount: tasks.length,

itemBuilder: (context, index) {

Task task = tasks[index];

return ListTile(

title: Text(task.title),

subtitle: Text(task.subtitle),

trailing: Checkbox(

value: task.isCompleted,

onChanged: (bool? value) {

task.isCompleted = value ?? false;

task.save();

},

),

);

},

);

},

),

);

}

}  
  
**Explanation:**

* The **HomeScreen** widget listens to changes in the tasksBox (Hive database).
* It displays a list of tasks using a ListView.builder. Each task is shown with a checkbox that toggles the isCompleted status.
* When a checkbox is clicked, the task’s completion status is updated in the database using task.save().

**2. Add Task: Use a Form to Add a New Task with Task Name and Due Date  
  
A screenshot of a phone

Description automatically generated**

**Logic:**  
Users can add new tasks by entering a title and description, along with a date and time.

**Code Explanation:  
import 'package:flutter/material.dart';**

**import 'package:intl/intl.dart';**

**import 'package:hive/hive.dart';**

**import '../models/task.dart';**

**import '../utils/strings.dart';**

**class AddTaskScreen extends StatefulWidget {**

**@override**

**\_AddTaskScreenState createState() => \_AddTaskScreenState();**

**}**

**class \_AddTaskScreenState extends State<AddTaskScreen> {**

**final TextEditingController titleController = TextEditingController();**

**final TextEditingController subtitleController = TextEditingController();**

**DateTime? selectedDate;**

**TimeOfDay? selectedTime;**

**void \_pickDate() async {**

**DateTime? date = await showDatePicker(**

**context: context,**

**initialDate: DateTime.now(),**

**firstDate: DateTime(2000),**

**lastDate: DateTime(2100),**

**);**

**if (date != null) {**

**setState(() {**

**selectedDate = date;**

**});**

**}**

**}**

**void \_pickTime() async {**

**TimeOfDay? time = await showTimePicker(**

**context: context,**

**initialTime: TimeOfDay.now(),**

**);**

**if (time != null) {**

**setState(() {**

**selectedTime = time;**

**});**

**}**

**}**

**void \_saveTask() {**

**if (titleController.text.isEmpty || subtitleController.text.isEmpty) {**

**ScaffoldMessenger.of(context).showSnackBar(**

**SnackBar(content: Text(MyString.emptyFieldsWarning)),**

**);**

**return;**

**}**

**final task = Task.create(**

**title: titleController.text,**

**subtitle: subtitleController.text,**

**createdAtTime: selectedTime != null**

**? DateTime(selectedDate!.year, selectedDate!.month, selectedDate!.day, selectedTime!.hour, selectedTime!.minute)**

**: DateTime.now(),**

**createdAtDate: selectedDate ?? DateTime.now(),**

**);**

**Hive.box<Task>('tasksBox').put(task.id, task);**

**Navigator.pop(context);**

**}**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(title: Text(MyString.addNewTask)),**

**body: Padding(**

**padding: const EdgeInsets.all(16.0),**

**child: Column(**

**children: [**

**TextField(**

**controller: titleController,**

**decoration: InputDecoration(labelText: MyString.titleOfTitleTextField),**

**),**

**TextField(**

**controller: subtitleController,**

**decoration: InputDecoration(labelText: MyString.addNote),**

**),**

**Row(**

**mainAxisAlignment: MainAxisAlignment.spaceBetween,**

**children: [**

**Text(selectedDate == null ? MyString.dateString : DateFormat.yMd().format(selectedDate!)),**

**IconButton(**

**icon: Icon(Icons.calendar\_today),**

**onPressed: \_pickDate,**

**),**

**],**

**),**

**Row(**

**mainAxisAlignment: MainAxisAlignment.spaceBetween,**

**children: [**

**Text(selectedTime == null ? MyString.timeString : selectedTime!.format(context)),**

**IconButton(**

**icon: Icon(Icons.access\_time),**

**onPressed: \_pickTime,**

**),**

**],**

**),**

**ElevatedButton(**

**onPressed: \_saveTask,**

**child: Text(MyString.addTaskString),**

**),**

**],**

**),**

**),**

**);**

**}**

**}  
  
  
Explanation:**

* Users input the task title and description via TextField widgets.
* They can pick a date and time using a DatePicker and TimePicker.
* The AddTaskScreen includes a validation to check if the fields are filled, and once saved, the task is stored in the Hive database.

**3. Mark Update/Delete: Update the UI Dynamically When Tasks Are Marked Complete or Deleted  
A screenshot of a chat

Description automatically generated**

**Logic:**  
When a user marks a task as completed, it should update the UI by crossing it off. Deleting a task removes it from the list.

**Code Explanation:**

void \_markAsComplete(Task task) {

task.isCompleted = true;

task.save(); // Updates the task status in the database

}

void \_deleteTask(Task task) {

task.delete(); // Removes the task from the database

}

* The task’s completion is marked using a checkbox on the task.
* A task is deleted by calling the delete() method from Hive.

The UI automatically updates when these operations are performed because of ValueListenableBuilder in the Home Screen, which listens to changes in the database.

**4. Filter Tasks**

**A screenshot of a social media post

Description automatically generated**

**Logic:**  
You can filter tasks into three categories:

* All
* Completed
* Pending

**Code Explanation:**

enum TaskStatus { all, completed, pending }

class HomeScreen extends StatefulWidget {

@override

\_HomeScreenState createState() => \_HomeScreenState();

}

class \_HomeScreenState extends State<HomeScreen> {

TaskStatus \_currentFilter = TaskStatus.all;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('To-Do List')),

body: Column(

children: [

Row(

children: [

TextButton(

onPressed: () => setState(() => \_currentFilter = TaskStatus.all),

child: Text('All'),

),

TextButton(

onPressed: () => setState(() => \_currentFilter = TaskStatus.completed),

child: Text('Completed'),

),

TextButton(

onPressed: () => setState(() => \_currentFilter = TaskStatus.pending),

child: Text('Pending'),

),

],

),

ValueListenableBuilder(

valueListenable: Hive.box<Task>('tasksBox').listenable(),

builder: (context, Box<Task> box, \_) {

var tasks = box.values.toList();

if (\_currentFilter == TaskStatus.completed) {

tasks = tasks.where((task) => task.isCompleted).toList();

} else if (\_currentFilter == TaskStatus.pending) {

tasks = tasks.where((task) => !task.isCompleted).toList();

}

return ListView.builder(

itemCount: tasks.length,

itemBuilder: (context, index) {

Task task = tasks[index];

return ListTile(

title: Text(task.title),

subtitle: Text(task.subtitle),

trailing: Checkbox(

value: task.isCompleted,

onChanged: (bool? value) {

task.isCompleted = value ?? false;

task.save();

},

),

);

},

);

},

),

],

),

);

}

}

**Explanation:**

* The filtering logic is implemented using the enum TaskStatus to switch between views based on the selected filter.
* When a filter is applied, the list of tasks is updated to show only tasks that match the filter criteria (completed, pending, or all).

**Features and Functionality**

1. **Home Screen**: The home screen is the central hub of the application, displaying all tasks in a list. Each task is shown with a checkbox to mark it as complete. The tasks are fetched from Hive’s local storage, ensuring that the data persists across sessions. The UI is dynamic, and any changes made (such as marking a task as complete or deleting it) are reflected in real time.
2. **Add Task**: Adding a task is simple. Users are presented with a form where they can enter the task name, description, and due date. The form also allows users to select a time for when the task is due. Once the user fills in the required information and presses "Add Task," the new task is stored in Hive, and the UI updates to reflect the changes.
3. **Mark Complete/Delete**: Users can mark a task as completed by checking the checkbox next to the task. The task’s status is updated in the Hive database, and the UI is immediately updated to reflect the change. Additionally, users can delete tasks they no longer need, and the task is removed from the database. These actions help keep the task list organized and current.
4. **Task Filtering**: The app supports task filtering, allowing users to view tasks based on their status. The options available are "All," "Completed," and "Pending." This feature helps users quickly access tasks that require attention, making it easier to stay organized and manage tasks based on their priority or completion status.
5. **Hive Integration**: The app utilizes Hive for local storage, which is a fast and lightweight NoSQL database. Tasks are saved in a Hive box, which allows for quick retrieval and manipulation of data. This ensures that the app is both performant and reliable, even as the user’s task list grows.

**Technology Stack**

* **Flutter**: The UI is built using Flutter, which enables cross-platform development for both Android and iOS.
* **Hive**: A lightweight and fast NoSQL database for Flutter that is used for local data storage.
* **Dart**: The programming language used for Flutter app development.
* **FTToast** and **Panara Dialogs**: For displaying notifications and dialog boxes.

**User Interface**

The user interface of the app is designed to be minimalistic and intuitive. The home screen lists tasks with clear labels for titles, descriptions, and due dates. The form to add a task is simple and clean, ensuring a user-friendly experience. Task filtering is implemented through buttons at the top of the screen, providing easy access to different task views.

**Conclusion**

This To-Do List application represents an innovative solution to daily task management, combining the powerful Flutter framework with the efficiency of Hive for local data storage. By offering features such as task creation, task completion, deletion, and filtering, the app helps users manage their time more effectively. The intuitive user interface and real-time updates enhance the user experience, making the app both functional and easy to use.

With this project, we have demonstrated how Flutter, coupled with local storage solutions like Hive, can be used to create a practical and efficient mobile application. This app not only provides a valuable tool for managing tasks but also showcases the power of Flutter in creating cross-platform mobile apps with minimal overhead.