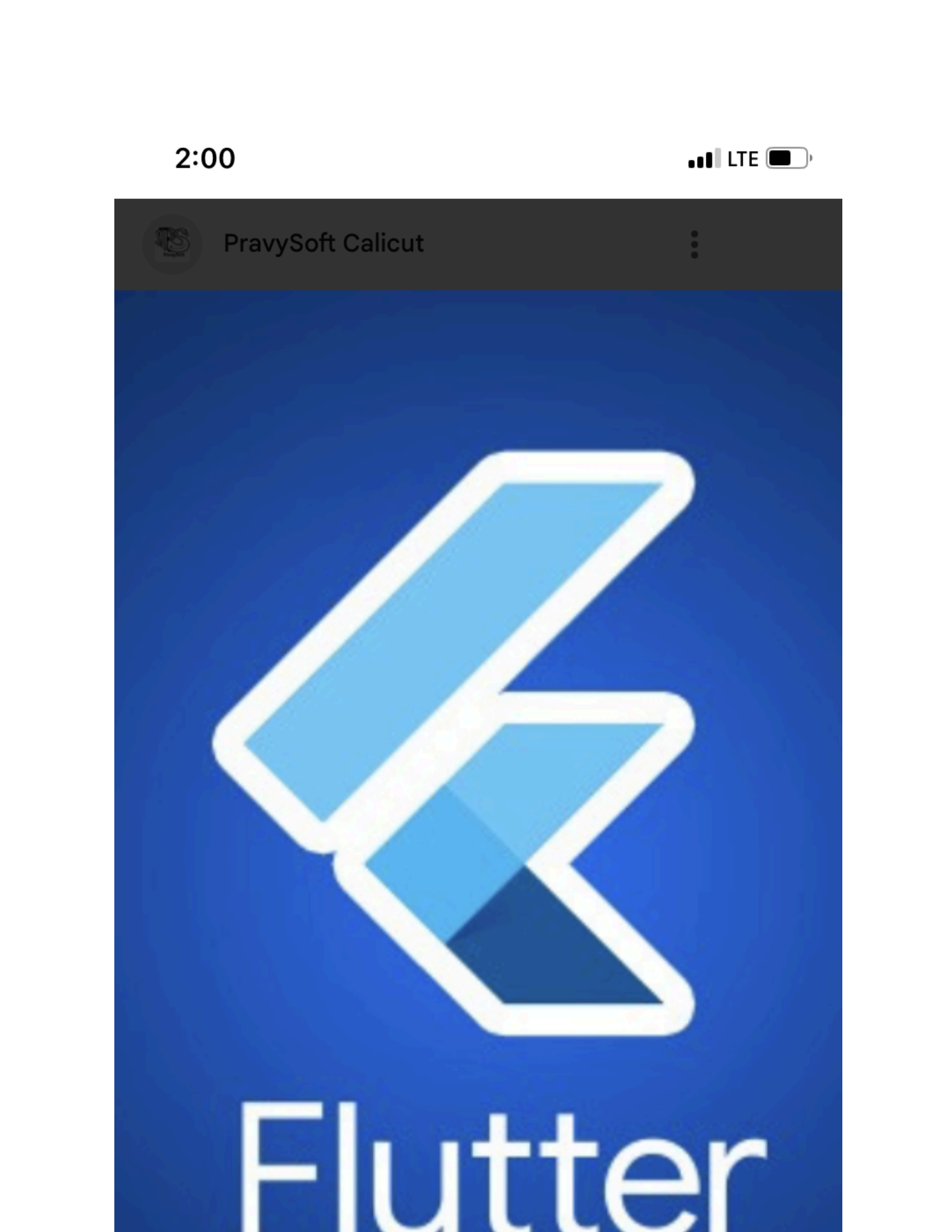
2:00
LTE
PravySoft Calicut
Flutter

Below is a beginner-friendly tutorial on how to create a simple Flutter app. We'll build a basic "To-Do List" app where users can add and remove tasks. This will help you understand how to set up a Flutter project, create UI components, and manage state.

Creating a Simple "To-Do List" App with Flutter

Step 1: Setting Up Flutter

Before we start coding, you need to set up Flutter on your machine.

1. **Download and Install Flutter**:

   * Go to the official [Flutter installation page](https://flutter.dev/docs/get-started/install).
   * Follow the instructions for your operating system (Windows, macOS, or Linux).

2. **Install Android Studio** (or any IDE):

   * You'll need an IDE to write and run your Flutter code. **Android Studio** is the most common choice, but you can also use **VS Code** with the Flutter plugin.
   * Download Android Studio from [here](https://developer.android.com/studio).

3. **Check Flutter Setup**:

   * Open your terminal/command prompt and run the following command to check if Flutter is installed correctly:

     ```bash
     flutter doctor
     ```
   * It will tell you if everything is set up, and if anything is missing, it will suggest steps to fix it.

Step 2: Create a New Flutter Project

Now that Flutter is installed, let's create our new project.

1. Open your terminal and run:

   ```bash
   flutter create todo_app
   ```

```
2. Navigate into your project directory:

```bash
cd todo_app
```
3. Open the project in your IDE (Android Studio or VS Code).

Step 3: Understanding the Default Project Structure

When you create a new Flutter project, it comes with some default files and folders:

* **lib/**: Contains the Dart code (where we'll write most of our app code).
* **ios/** & **android/**: These contain platform-specific code for iOS and Android (you don't need to modify these for now).
* **pubspec.yaml**: Contains dependencies and metadata for the project.

We will mainly work in the `lib/` folder.

Step 4: Set Up the Main App Structure

1. In the `lib/` folder, open `main.dart` and delete the default code.

2. Replace it with the following:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'To-Do List App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
```
```

```
      home: TodoScreen(),
    );
  }
}

class TodoScreen extends StatefulWidget {
  @override
  _TodoScreenState createState() => _TodoScreenState();
}

class _TodoScreenState extends State<TodoScreen> {
  List<String> tasks = [];
  TextEditingController taskController = TextEditingController();

  void addTask() {
    if (taskController.text.isNotEmpty) {
      setState(() {
        tasks.add(taskController.text);
      });
      taskController.clear();
    }
  }

  void removeTask(int index) {
    setState(() {
      tasks.removeAt(index);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('To-Do List'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            TextField(
              controller: taskController,
              decoration: InputDecoration(
                labelText: 'Enter a task',
                border: OutlineInputBorder(),
```

```
            ),
          ),
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: addTask,
            child: Text('Add Task'),
          ),
          SizedBox(height: 20),
          Expanded(
            child: ListView.builder(
              itemCount: tasks.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(tasks[index]),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () => removeTask(index),
                  ),
                );
              },
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

Step 5: How the Code Works

Let’s break down the code we just wrote:

1. **MyApp Class**:

   * This is the root of our application. The `MaterialApp` widget is used to set up basic app-wide configurations like theme and title.

2. **TodoScreen Class**:

   * This is the main screen of our app. It contains a stateful widget because the to-do list will change over time (tasks get added and removed).

3. **State Management**:

   * **tasks**: A list that holds the to-do items.
   * **taskController**: A controller for the text input field, so we can retrieve the text the user types.
   * **addTask()**: Adds a new task to the `tasks` list and clears the input field.
   * **removeTask(index)**: Removes a task from the list when the delete button is pressed.

4. **UI Components**:

   * We use a `TextField` to take input from the user.
   * An `ElevatedButton` allows the user to add tasks.
   * A `ListView.builder` is used to display the list of tasks, and each task has a delete button (using `IconButton`).

Step 6: Run the App

1. **Start an Emulator**:

   * If you're using Android Studio, start an emulator or connect your physical device.

2. **Run the App**:

   * In your terminal, run the command:

     ```bash
     flutter run
     ```

3. **Test the App**:

   * Once the app launches, you should see a text input field where you can type a task and a button to add it.
   * The tasks will appear below, and you can delete them using the trash icon next to each task.

Step 7: Enhancements (Optional)

1. **Persistent Storage**:

   * Right now, tasks disappear when the app closes. You can use a package like `shared_preferences` to store tasks persistently.

2. **Custom UI**:

   * You can customize the UI further by adding colors, fonts, or animations to make the app look more appealing.

3. **Add Task Validation**:

   * You could add logic to prevent empty tasks from being added.

Conclusion

Congratulations! You've just built your first Flutter app. 🎉

* We covered setting up the Flutter environment, creating a simple to-do app, and managing state.
* Flutter’s **rich set of widgets**, **hot reload**, and **easy-to-learn Dart** language make it a great choice for building cross-platform mobile apps quickly.

Feel free to expand this project, try adding more features like task editing, or improve the UI with Flutter's built-in design elements. If you have any questions or need further help, don’t hesitate to ask!

What’s next? Would you like to try adding persistent storage, or do you want to explore another feature of Flutter?