Module 2 (Part 2): Dynamic UIs with Models and `ListView.builder`

In Part 1, we mastered building static UIs with hard-coded data. While our apps looked good, the code was inflexible. To make any changes, we had to edit the UI code directly. In this module, we will make the most important architectural leap in app development: we will separate our **data** (the "what") from our **UI** (the "how"). We will create dedicated data "model" classes, store our mock data in separate files, and use the powerful `ListView.builder` to dynamically render our UI based on that clean data source.

1. Learning Objectives

- Understand and articulate the critical importance of "Separation of Concerns" (data vs. UI).
- Create dedicated Dart "model" classes to define the structure of application data (e.g., a `Product` or a `Task`).
- Organize a project by creating separate files for models and mock data sources.
- Master the use of `ListView.builder` as the efficient, professional way to build dynamic lists.
- Understand and implement the `itemCount` and `itemBuilder` properties.
- Pass entire data objects to widgets, making the UI a "template" for the data.
- Refactor both projects to be fully data-driven, scalable, and maintainable.

2. Core Concepts

A. Fundamental Theory: Models and Separation of Concerns

The Problem: In Part 1, our data (like `'Classic Leather Jacket'` or `'$149.99'`) was directly embedded within our UI widgets. This is a maintenance nightmare. To add a new product, you have to copy-paste a large chunk of UI code. This is brittle, error-prone, and doesn't scale.

The Solution: Model Classes.
A model class is a pure Dart class that acts as a blueprint for our data. It contains no UI code—its only job is to define the properties of a real-world object.

```
// A blueprint for a Product. No UI, just data structure.
class Product {
  final String id;
  final String name;
  final double price;
  final String imageUrl;
  // Constructor to create instances of a Product
```

```
  Product({required this.id, required this.name, required this.price,
required this.imageUrl});
}
```

By creating a `List<Product>` in its own file, we achieve **Separation of Concerns**. Our UI files are only concerned with *how to display* a product. Our data files are only concerned with *what a product is*. This is the foundation of all professional app architecture.

B. Exploration of Options (Comparative Analysis): Building Dynamic Lists

- **Option 1: `ListView` with a `List<Widget>` (The Eager Constructor)**
    - **How:** You create a list of widgets in memory first, then pass it to the `ListView`.
    - **Why it's bad:** It's "eager." It builds **every single widget** in your list at once, even if you have 10,000 items and only 5 are visible. This will crash your app with high memory usage. It is only acceptable for a tiny, fixed number of items.
- **Option 2: `ListView.builder` (The Lazy, Professional Constructor)**
    - **How:** You provide two things: `itemCount` (how many items in total) and an `itemBuilder` (a recipe function that tells Flutter how to build *one* item at a given position).
    - **Why it's great:** It's "lazy." It only calls the `itemBuilder` function for the items that are currently visible on the screen. As you scroll, it recycles the widgets that go off-screen and reuses them for the new ones coming into view. This keeps memory usage low and performance high, no matter how long your list is.

C. Our Chosen Approach & Rationale

We will exclusively use `ListView.builder` for all dynamic lists in our projects.

Rationale: `ListView.builder` is not just an option; it is the professional standard for building lists from a data source. It forces us to adopt the "Separation of Concerns" principle and guarantees that our applications are performant and scalable. Using the simple `ListView` constructor for a dynamic list is a common beginner mistake that we will avoid from the start.

3. Practical Application: ShopSphere

Goal: Transform our static list of product cards into a dynamic, scrollable list driven by a clean, separate mock data source.

Step-by-Step Implementation:

1. **Create Model File:** In the `lib` folder, create a new folder `models/`. Inside, create `product.dart` and define the `Product` class.

2. **Create Data File:** In `lib`, create a new folder `data/`. Inside, create `mock_data.dart` and define the `final List<Product> mockProducts`.
3. **Refactor `ProductCard`:** Modify the `ProductCard` widget to accept a single `final Product product;` object in its constructor instead of separate strings.
4. **Refactor `HomePage`:** Replace the old `ListView` with `ListView.builder`, pointing it to the imported `mockProducts` list and using the `itemBuilder` to create a `ProductCard` for each product.

4. Practical Application: TaskFlow

Goal: Apply the exact same data-driven architecture to our task list.

Step-by-Step Implementation:

1. **Create Model File:** In `task_flow/lib`, create `models/task.dart` and define the `Task` class (with `id`, `title`, `dueDate`, `isCompleted`).
2. **Create Data File:** In `task_flow/lib`, create `data/mock_data.dart` and define the `final List<Task> mockTasks`.
3. **Refactor `TaskItem`:** This is a crucial step. The `TaskItem` will now be driven by a `Task` object. We will remove the internal `_isChecked` state from the widget, as the `task.isCompleted` property is now the single source of truth.
4. **Refactor `HomePage`:** Replace the hard-coded `ListView` with a `ListView.builder` that iterates over the `mockTasks` list.

5. Full Code for New/Updated Files

`shop_sphere/lib/models/product.dart` (New)

```
class Product {
  final String id;
  final String name;
  final double price;
  final String imageUrl;

  const Product({
    required this.id,
    required this.name,
    required this.price,
    required this.imageUrl,
  });
}
```

shop_sphere/lib/data/mock_data.dart (New)

```
import 'package:shop_sphere/models/product.dart';

// This file is now the single source of truth for our mock product data.
final List<Product> mockProducts = [
  const Product(id: '1', name: 'Modern Accent Chair', price: 299.99,
imageUrl: 'https://picsum.photos/seed/chair/600/400'),
  const Product(id: '2', name: 'Minimalist Desk Lamp', price: 79.50,
imageUrl: 'https://picsum.photos/seed/lamp/600/400'),
  const Product(id: '3', name: 'Hand-Woven Area Rug', price: 450.00,
imageUrl: 'https://picsum.photos/seed/rug/600/400'),
  const Product(id: '4', name: 'Smart Coffee Maker', price: 125.99, imageUrl:
'https://picsum.photos/seed/coffee/600/400'),
  const Product(id: '5', name: 'Noise-Cancelling Headphones', price: 349.00,
imageUrl: 'https://picsum.photos/seed/headphones/600/400'),
  const Product(id: '6', name: 'Ergonomic Office Chair', price: 499.99,
imageUrl: 'https://picsum.photos/seed/officechair/600/400'),
  const Product(id: '7', name: 'Vintage Bluetooth Speaker', price: 199.00,
imageUrl: 'https://picsum.photos/seed/speaker/600/400'),
];
```

shop_sphere/lib/widgets/product_card.dart (Refactored)

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

class ProductCard extends StatelessWidget {
  // We now accept a single Product object. This makes our widget signature
clean.
  final Product product;

  const ProductCard({super.key, required this.product});

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 4.0,
      margin: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
```

```
        children: [
          ClipRRect(
            borderRadius: const BorderRadius.only(
              topLeft: Radius.circular(12.0),
              topRight: Radius.circular(12.0),
            ),
            child: Image.network(
              product.imageUrl, // Use data from the model object
              height: 200,
              width: double.infinity,
              fit: BoxFit.cover,
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  product.name, // Use data from the model object
                  style: const TextStyle(fontSize: 20, fontWeight:
FontWeight.bold),
                  maxLines: 1,
                  overflow: TextOverflow.ellipsis,
                ),
                const SizedBox(height: 8.0),
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Text(
                      // Format the double price to a string with a dollar
sign
                      '\$${product.price.toStringAsFixed(2)}',
                      style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.w600,
                        color: Theme.of(context).colorScheme.primary,
                      ),
                    ),
```

```
                    ElevatedButton(
                      onPressed: () {},
                      child: const Text('Add to Cart'),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

shop_sphere/lib/home_page.dart

```
import 'package:flutter/material.dart';
import 'package:shop_sphere/data/mock_data.dart';
import 'package:shop_sphere/widgets/product_card.dart';

class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('ShopSphere'),
        actions: [
          IconButton(
            icon: const Icon(Icons.shopping_cart_outlined),
            onPressed: () {},
          ),
        ],
      ),
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Padding(
            padding: EdgeInsets.fromLTRB(16.0, 16.0, 16.0, 8.0),
```

```
            child: Text(
              'Featured Products',
              style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
            ),
          ),
          Expanded(
            // We replace the hard-coded list with ListView.builder
            child: ListView.builder(
              // 1. Provide the total number of items in the list.
              itemCount: mockProducts.length,
              // 2. Provide the builder function. This function will be
called
              // for each item, from index 0 to itemCount - 1.
              itemBuilder: (BuildContext context, int index) {
                // Get the specific product for this index.
                final product = mockProducts[index];
                // Return the widget for this item.
                return ProductCard(product: product);
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

`task_flow/lib/models/task.dart` (New)

```
class Task {
  final String id;
  final String title;
  final String dueDate;
  bool isCompleted; // This is not final, as its value needs to change.

  Task({
    required this.id,
    required this.title,
    required this.dueDate,
```

```
    this.isCompleted = false,
  });
}
```

task_flow/lib/data/mock_data.dart (New)

```
import 'package:task_flow/models/task.dart';

// This file now contains our mock task data.
final List<Task> mockTasks = [
  Task(id: '1', title: 'Finish Module 2 of Flutter Course', dueDate:
'Today'),
  Task(id: '2', title: 'Submit the weekly report', dueDate: 'Today'),
  Task(id: '3', title: 'Call the client back', dueDate: 'Tomorrow',
isCompleted: true),
  Task(id: '4', title: 'Grocery shopping for the week', dueDate: 'Oct 28'),
  Task(id: '5', title: 'Plan the weekend trip', dueDate: 'Oct 29'),
  Task(id: '6', title: 'Read one chapter of "Atomic Habits"', dueDate:
'Today'),
  Task(id: '7', title: 'Fix bug #734 on the main project', dueDate:
'Tomorrow', isCompleted: true),
];
```

task_flow/lib/task_item.dart (Refactored)

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

class TaskItem extends StatefulWidget {
  final Task task;
  const TaskItem({super.key, required this.task});

  @override
  State<TaskItem> createState() => _TaskItemState();
}

class _TaskItemState extends State<TaskItem> {
  // This local state is for immediate UI feedback ONLY.
  // It is initialized from the "true" data source, `widget.task`.
  late bool _isChecked;

  @override
  void initState() {
```

```
    super.initState();
    _isChecked = widget.task.isCompleted;
  }

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 4.0),
      child: ListTile(
        leading: Checkbox(
          value: _isChecked,
          onChanged: (value) {
            setState(() {
              _isChecked = value ?? false;
            });
            // IMPORTANT NOTE FOR STUDENTS:
            // This only changes the UI visually for a moment.
            // The underlying data in the `mockTasks` list is NOT being
updated.
            // If you scroll this item off-screen and back, it will revert.
            // We will solve this with proper State Management in a future
module.
          },
        ),
        title: Text(
          widget.task.title,
          style: TextStyle(
            decoration:
                _isChecked ? TextDecoration.lineThrough :
TextDecoration.none,
            color: _isChecked ? Colors.grey[600] : null,
          ),
        ),
        subtitle: Text('Due: ${widget.task.dueDate}'),
        trailing: IconButton(
          icon: const Icon(Icons.delete_outline, color: Colors.redAccent),
          onPressed: () {
            // Delete logic will be added later
          },
```

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

`task_flow/lib/home_page.dart` (Refactored)

```
import 'package:flutter/material.dart';
import 'package:task_flow/data/mock_data.dart';
import 'package:task_flow/task_item.dart';

class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('My Tasks'),
      ),
      body: ListView.builder(
        // The itemCount is now dynamically set from the length of our data
list.
        itemCount: mockTasks.length,
        itemBuilder: (context, index) {
          // For each index, get the corresponding task from our data list.
          final task = mockTasks[index];
          // Return the TaskItem widget, passing the task data to it.
          // Providing a unique Key is crucial for lists that can change,
          // helping Flutter efficiently update the UI.
          return TaskItem(key: ValueKey(task.id), task: task);
        },
      ),
    );
  }
}
```

6. Assignment/Challenge

1. **For `ShopSphere`:** Add a new boolean property to your `Product` model called `onSale` and a `double?` property called `salePrice`. In your `mock_data.dart`, set a few products to be on sale. In `product_card.dart`, use an `if` statement to conditionally display the `salePrice` with a strikethrough on the original `price`.
2. **For `TaskFlow`:** In your `Task` model, change the `dueDate` from a `String` to Dart's built-in `DateTime` object. This is a much better data type for dates. You will need to update your mock data (`DateTime.now()`) and use a package like `intl` to format the `DateTime` object back into a readable string in the `TaskItem` widget.
3. **GridView:** `ShopSphere` might look better with a grid layout instead of a list. In `home_page.dart`, replace `ListView.builder` with `GridView.builder`. You will need to provide a `gridDelegate`. A good one to start with is `SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2)`. Observe how easily you can switch between list and grid layouts when your UI is properly separated from your data.