Exercício de Refatoração com DDD no Flutter

Cenário Inicial: O "God Widget"

Imagine que você recebeu um projeto simples de uma lista de afazeres (To-Do List) onde **toda a lógica** de negócio, acesso a dados e UI está contida em um único StatefulWidget.

Arquivo: lib/main.dart

Dart

```
// lib/main.dart (CÓDIGO INICIAL A SER REFATORADO)

import 'package:flutter/material.dart';


class TodoItem {
  final String title;
  bool isCompleted;


  TodoItem(this.title, {this.isCompleted = false});
}


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


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


  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'DDD Refactor Challenge',
      home: TodoListScreen(),
    );
  }
}
```

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

  @override
  State<TodoListScreen> createState() => _TodoListScreenState();
}

class _TodoListScreenState extends State<TodoListScreen> {
  final List<TodoItem> _todos = [];
  final TextEditingController _controller = TextEditingController();

  // Lógica de "Serviço/Infraestrutura"
  void _loadTodosFromStorage() {
    // Simula o carregamento de dados (Infraestrutura)
    setState(() {
      _todos.add(TodoItem('Comprar leite', isCompleted: false));
      _todos.add(TodoItem('Estudar DDD', isCompleted: true));
    });
  }

  // Lógica de Negócio (Domínio)
  void _addTodo() {
    final title = _controller.text.trim();
    if (title.isNotEmpty && title.length < 50) { // Regra de Negócio: limite de 50 caracteres
      setState(() {
        _todos.add(TodoItem(title));
        _controller.clear();
      });
    } else if (title.length >= 50) {
      // Simulação de erro/feedback da UI
```

```
   ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('A tarefa é muito longa!')),
   );
  }
}

// Lógica de Negócio (Domínio)
void _toggleTodo(int index) {
  setState(() {
   _todos[index].isCompleted = !_todos[index].isCompleted;
  });
}

@override
void initState() {
  super.initState();
  _loadTodosFromStorage();
}

@override
Widget build(BuildContext context) {
  // Lógica de Apresentação/UI
  return Scaffold(
   appBar: AppBar(title: const Text('Minhas Tarefas (God Widget)')),
   body: Column(
    children: [
     Padding(
      padding: const EdgeInsets.all(8.0),
      child: Row(
       children: [
        Expanded(
```

```
      child: TextField(
        controller: _controller,
        decoration: const InputDecoration(labelText: 'Nova Tarefa'),
      ),
    ),
    ElevatedButton(
      onPressed: _addTodo,
      child: const Text('Adicionar'),
    ),
  ],
 ),
),
Expanded(
 child: ListView.builder(
  itemCount: _todos.length,
  itemBuilder: (context, index) {
   final todo = _todos[index];
   return ListTile(
    title: Text(
     todo.title,
     style: TextStyle(
      decoration: todo.isCompleted ? TextDecoration.lineThrough : null,
     ),
    ),
    trailing: Checkbox(
     value: todo.isCompleted,
     onChanged: (_) => _toggleTodo(index),
    ),
   );
  },
 ),
```

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

O Desafio: Aplicar DDD (Clean Architecture)

Seu objetivo é refatorar o código acima, movendo a lógica para as camadas apropriadas e separando a aplicação em um **Contexto Delimitado** chamado **todo**.

Requisitos de Refatoração

Crie a seguinte estrutura de diretórios e mova os blocos de lógica conforme a filosofia DDD:

1. Camada de Domínio (lib/features/todo/domain/)

- **Value Object:** Crie um **TodoTitle** que encapsule o título da tarefa.
    - **Regra de Negócio:** Deve validar que o título **não está vazio** e tem no máximo **50 caracteres**. (Substitua a validação na UI/Widget).
- **Entidade:** Refatore a classe TodoItem para ser uma **Entidade** no Domínio, garantindo que seu estado seja consistente (imutabilidade com métodos de cópia, se possível). Renomeie-a para **Todo**.
- **Interface de Repositório:** Crie a interface **ITodoRepository** com os métodos: getTodos(), saveTodo(Todo todo), e updateTodo(Todo todo).
- **Casos de Uso (Use Cases):** Crie os Casos de Uso que interagem com o Repositório:
    - GetTodos
    - AddTodo
    - ToggleTodoStatus

2. Camada de Infraestrutura/Dados (lib/features/todo/data/)

- **Data Source:** Crie um **TodoLocalDataSource** que simula o acesso a dados (mantendo a lista _todos internamente ou usando uma lista estática temporária).
- **Implementação do Repositório:** Crie a classe **TodoRepositoryImpl** que implementa ITodoRepository e usa o TodoLocalDataSource para obter e salvar os dados.

3. Camada de Apresentação (lib/features/todo/presentation/)

- **Gerenciamento de Estado:** Use um ChangeNotifier (ou BLoC/Cubit, se souber usar) chamado **TodoNotifier** (ou TodoCubit) para gerenciar a lista de tarefas (List<Todo>).
    - Este Notifier **deve depender e chamar apenas os Casos de Uso** do Domínio.
- **UI:** Refatore a TodoListScreen para ser um StatelessWidget (ou um *Consumer* do seu gerenciador de estado) que apenas **observa** o estado do TodoNotifier e chama seus métodos (que, por sua vez, chamam os Casos de Uso).

Estrutura de Diretórios Esperada

O resultado final deve se encaixar na seguinte estrutura:

```
lib/
├── features/
│   └── todo/
│       ├── domain/
│       │   ├── entities/
│       │   │   └── todo.dart          # Entidade Todo
│       │   │   └── todo_title.dart    # Value Object TodoTitle
│       │   ├── repositories/
│       │   │   └── i_todo_repository.dart # Interface
│       │   └── usecases/
│       │       └── add_todo.dart      # Caso de Uso
│       │       └── get_todos.dart     # Caso de Uso
│       │       └── toggle_todo_status.dart # Caso de Uso
│       ├── data/
│       │   ├── datasources/
│       │   │   └── todo_local_datasource.dart # Simulação de Infra
│       │   └── repositories/
│       │       └── todo_repository_impl.dart # Implementação da Interface
│       └── presentation/
│           ├── notifiers/
│           │   └── todo_notifier.dart   # Gerenciamento de Estado (Usa Use Cases)
│           └── pages/
```

│ └── todo_list_screen.dart # Stateless UI (Observa o Notifier)

└── main.dart # Apenas inicializa o app e injeta dependências (se for o caso)

Critérios de Avaliação

1. **Isolamento do Domínio:** A camada domain deve ser totalmente independente, sem *nenhum* import de data, presentation ou flutter:material.
2. **Encapsulamento:** A validação do limite de 50 caracteres deve ser realizada pelo **TodoTitle** (Value Object).
3. **Separação de Preocupações:** O TodoListScreen não deve mais conter nenhuma lógica de negócio ou acesso a dados. Ele apenas chama métodos do TodoNotifier.
4. **Funcionalidade:** O aplicativo deve continuar funcionando exatamente como o original (adicionar, marcar/desmarcar).

Dica: Para facilitar a refatoração e a injeção de dependência na camada de presentation, você pode usar o pacote provider ou riverpod (se tiver familiaridade) ou, de forma mais simples, injetar as dependências no construtor do TodoNotifier e usar o get_it no main.dart para registrar as implementações.