Mobile UI e UX melhorada

- Vencimento 2 nov por 23:59
- Pontos 0
- Disponível até 2 nov em 23:59

Esta tarefa foi travada 2 nov em 23:59.

Aula 2: UI Melhorada e Navegação entre Telas

Laboratório de Desenvolvimento de Aplicações Móveis e Distribuídas
Curso de Engenharia de Software - PUC Minas
Duração: 1h30m (20min teoria + 70min prática)

📋 Objetivos da Aula

Ao final desta aula, você será capaz de:

- ✅ Criar interface seguindo Material Design 3
- ✅ Implementar navegação entre múltiplas telas
- ✅ Desenvolver formulários completos com validação
- ✅ Aplicar boas práticas de UX mobile
- ✅ Gerenciar estado entre telas diferentes

📚 Conteúdo Teórico (20 minutos)

1. Material Design 3 no Flutter

O que é Material Design 3?

Material Design 3 (também conhecido como Material You) é a terceira geração do sistema de design do Google, focado em:

- **Personalização**: Temas dinâmicos baseados em cores
- **Expressividade**: Componentes mais flexíveis e personalizáveis
- **Acessibilidade**: Contraste aprimorado e melhor usabilidade

Principais Componentes:

```
┌─────────────────────────────────────┐
│   Material Design 3 Components      │
├─────────────────────────────────────┤
│  • Cards elevados e contornados     │
│  • Floating Action Buttons (FAB)    │
│  • NavigationBar e NavigationRail   │
│  • TextField com outline/filled     │
```

```
│   • SnackBars e Dialogs                 │
│   • Chips, Badges e Icons               │
└─────────────────────────────────────────┘
```

2. Navegação no Flutter

Stack de Navegação:

```
┌──────────────────────┐
│   Tela C (Topo)      │  ← Navigator.push()
├──────────────────────┤
│   Tela B             │  ← Navigator.pop()
├──────────────────────┤
│   Tela A (Base)      │
└──────────────────────┘
```

Tipos de Navegação:

Tipo	Uso	Método
Push	Adicionar tela	`Navigator.push()`
Pop	Voltar	`Navigator.pop()`
Replace	Substituir	`Navigator.pushReplacement()`
Named Routes	Rotas nomeadas	`Navigator.pushNamed()`

3. Formulários e Validação

Componentes de Formulário:

```
Form
├── TextFormField (título)
├── TextFormField (descrição)
├── DropdownButtonFormField (prioridade)
├── SwitchListTile (completo)
└── ElevatedButton (salvar)
```

Validação em Tempo Real:

```
GlobalKey<FormState> _formKey = GlobalKey<FormState>();

if (_formKey.currentState!.validate()) {
  // Dados válidos, pode salvar
  _formKey.currentState!.save();
}
```

4. Princípios de UX Mobile

Regras de Ouro:

1. **Área de Toque**: Mínimo 48x48 pixels
2. **Hierarquia Visual**: Tamanhos e cores indicam importância
3. **Feedback Imediato**: Loading, animações, confirmações
4. **Consistência**: Padrões uniformes em toda a app
5. **Acessibilidade**: Contraste, tamanho de fonte, labels

💻 Prática (70 minutos)

PASSO 1: Preparar Projeto Base

1.1 Continuar do Laboratório Anterior

```
cd task_manager
```

Se não tiver o projeto do Lab 1, clone ou recrie seguindo o roteiro anterior.

1.2 Verificar Estrutura Atual

```
task_manager/
├── lib/
│   ├── main.dart
│   ├── models/
│   │   └── task.dart
│   ├── services/
│   │   └── database_service.dart
│   └── screens/
│       └── task_list_screen.dart  (existente)
└── pubspec.yaml
```

PASSO 2: Criar Tela de Formulário

2.1 Criar Arquivo `lib/screens/task_form_screen.dart`

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

class TaskFormScreen extends StatefulWidget {
  final Task? task; // null = criar novo, não-null = editar

  const TaskFormScreen({super.key, this.task});

  @override
  State<TaskFormScreen> createState() => _TaskFormScreenState();
}

class _TaskFormScreenState extends State<TaskFormScreen> {
  final _formKey = GlobalKey<FormState>();
  final _titleController = TextEditingController();
  final _descriptionController = TextEditingController();

  String _priority = 'medium';
  bool _completed = false;
  bool _isLoading = false;

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

    // Se estiver editando, preencher campos
    if (widget.task != null) {
      _titleController.text = widget.task!.title;
      _descriptionController.text = widget.task!.description;
      _priority = widget.task!.priority;
      _completed = widget.task!.completed;
    }
```

```
  }

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

  Future<void> _saveTask() async {
    if (!_formKey.currentState!.validate()) {
      return;
    }

    setState(() => _isLoading = true);

    try {
      if (widget.task == null) {
        // Criar nova tarefa
        final newTask = Task(
          title: _titleController.text.trim(),
          description: _descriptionController.text.trim(),
          priority: _priority,
          completed: _completed,
        );
        await DatabaseService.instance.create(newTask);

        if (mounted) {
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
              content: Text('✓ Tarefa criada com sucesso'),
              backgroundColor: Colors.green,
              duration: Duration(seconds: 2),
            ),
          );
        }
      } else {
        // Atualizar tarefa existente
        final updatedTask = widget.task!.copyWith(
          title: _titleController.text.trim(),
          description: _descriptionController.text.trim(),
          priority: _priority,
          completed: _completed,
        );
        await DatabaseService.instance.update(updatedTask);

        if (mounted) {
          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
              content: Text('✓ Tarefa atualizada com sucesso'),
              backgroundColor: Colors.blue,
              duration: Duration(seconds: 2),
            ),
          );
        }
      }

      if (mounted) {
        Navigator.pop(context, true); // Retorna true = sucesso
      }
    } catch (e) {
      if (mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('Erro ao salvar: $e'),
            backgroundColor: Colors.red,
          ),
        );
      }
    } finally {
      if (mounted) {
```

```
      setState(() => _isLoading = false);
    }
  }
}

@override
Widget build(BuildContext context) {
  final isEditing = widget.task != null;

  return Scaffold(
    appBar: AppBar(
      title: Text(isEditing ? 'Editar Tarefa' : 'Nova Tarefa'),
      backgroundColor: Colors.blue,
      foregroundColor: Colors.white,
    ),
    body: _isLoading
        ? const Center(child: CircularProgressIndicator())
        : SingleChildScrollView(
            padding: const EdgeInsets.all(16),
            child: Form(
              key: _formKey,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: [
                  // Campo de Título
                  TextFormField(
                    controller: _titleController,
                    decoration: const InputDecoration(
                      labelText: 'Título *',
                      hintText: 'Ex: Estudar Flutter',
                      prefixIcon: Icon(Icons.title),
                      border: OutlineInputBorder(),
                    ),
                    textCapitalization: TextCapitalization.sentences,
                    validator: (value) {
                      if (value == null || value.trim().isEmpty) {
                        return 'Por favor, digite um título';
                      }
                      if (value.trim().length < 3) {
                        return 'Título deve ter pelo menos 3 caracteres';
                      }
                      return null;
                    },
                    maxLength: 100,
                  ),

                  const SizedBox(height: 16),

                  // Campo de Descrição
                  TextFormField(
                    controller: _descriptionController,
                    decoration: const InputDecoration(
                      labelText: 'Descrição',
                      hintText: 'Adicione mais detalhes...',
                      prefixIcon: Icon(Icons.description),
                      border: OutlineInputBorder(),
                      alignLabelWithHint: true,
                    ),
                    textCapitalization: TextCapitalization.sentences,
                    maxLines: 5,
                    maxLength: 500,
                  ),

                  const SizedBox(height: 16),

                  // Dropdown de Prioridade
                  DropdownButtonFormField<String>(
                    decoration: const InputDecoration(
                      labelText: 'Prioridade',
                      prefixIcon: Icon(Icons.flag),
                      border: OutlineInputBorder(),
```

```
    ),
    items: const [
      DropdownMenuItem(
        value: 'low',
        child: Row(
          children: [
            Icon(Icons.flag, color: Colors.green),
            SizedBox(width: 8),
            Text('Baixa'),
          ],
        ),
      ),
      DropdownMenuItem(
        value: 'medium',
        child: Row(
          children: [
            Icon(Icons.flag, color: Colors.orange),
            SizedBox(width: 8),
            Text('Média'),
          ],
        ),
      ),
      DropdownMenuItem(
        value: 'high',
        child: Row(
          children: [
            Icon(Icons.flag, color: Colors.red),
            SizedBox(width: 8),
            Text('Alta'),
          ],
        ),
      ),
      DropdownMenuItem(
        value: 'urgent',
        child: Row(
          children: [
            Icon(Icons.flag, color: Colors.purple),
            SizedBox(width: 8),
            Text('Urgente'),
          ],
        ),
      ),
    ],
    onChanged: (value) {
      if (value != null) {
        setState(() => _priority = value);
      }
    },
  ),

  const SizedBox(height: 16),

  // Switch de Completo
  Card(
    child: SwitchListTile(
      title: const Text('Tarefa Completa'),
      subtitle: Text(
        _completed
            ? 'Esta tarefa está marcada como concluída'
            : 'Esta tarefa ainda não foi concluída',
      ),
      value: _completed,
      onChanged: (value) {
        setState(() => _completed = value);
      },
      secondary: Icon(
        _completed ? Icons.check_circle : Icons.radio_button_unchecked,
        color: _completed ? Colors.green : Colors.grey,
      ),
    ),
  ),
```

```
                    const SizedBox(height: 24),

                    // Botão Salvar
                    ElevatedButton.icon(
                      onPressed: _saveTask,
                      icon: const Icon(Icons.save),
                      label: Text(isEditing ? 'Atualizar Tarefa' : 'Criar Tarefa'),
                      style: ElevatedButton.styleFrom(
                        padding: const EdgeInsets.all(16),
                        backgroundColor: Colors.blue,
                        foregroundColor: Colors.white,
                        shape: RoundedRectangleBorder(
                          borderRadius: BorderRadius.circular(8),
                        ),
                      ),
                    ),

                    const SizedBox(height: 8),

                    // Botão Cancelar
                    OutlinedButton.icon(
                      onPressed: () => Navigator.pop(context),
                      icon: const Icon(Icons.cancel),
                      label: const Text('Cancelar'),
                      style: OutlinedButton.styleFrom(
                        padding: const EdgeInsets.all(16),
                        shape: RoundedRectangleBorder(
                          borderRadius: BorderRadius.circular(8),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
    );
  }
}
```

PASSO 3: Criar Widget de Card para Tarefa

3.1 Criar Arquivo `lib/widgets/task_card.dart`

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

class TaskCard extends StatelessWidget {
  final Task task;
  final VoidCallback onTap;
  final VoidCallback onToggle;
  final VoidCallback onDelete;

  const TaskCard({
    super.key,
    required this.task,
    required this.onTap,
    required this.onToggle,
    required this.onDelete,
  });

  Color _getPriorityColor() {
    switch (task.priority) {
      case 'low':
        return Colors.green;
```

```
      case 'medium':
        return Colors.orange;
      case 'high':
        return Colors.red;
      case 'urgent':
        return Colors.purple;
      default:
        return Colors.grey;
    }
  }

  IconData _getPriorityIcon() {
    switch (task.priority) {
      case 'urgent':
        return Icons.priority_high;
      default:
        return Icons.flag;
    }
  }

  String _getPriorityLabel() {
    switch (task.priority) {
      case 'low':
        return 'Baixa';
      case 'medium':
        return 'Média';
      case 'high':
        return 'Alta';
      case 'urgent':
        return 'Urgente';
      default:
        return 'Média';
    }
  }

  @override
  Widget build(BuildContext context) {
    final dateFormat = DateFormat('dd/MM/yyyy HH:mm');

    return Card(
      margin: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      elevation: task.completed ? 1 : 3,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12),
        side: BorderSide(
          color: task.completed ? Colors.grey.shade300 : _getPriorityColor(),
          width: 2,
        ),
      ),
      child: InkWell(
        onTap: onTap,
        borderRadius: BorderRadius.circular(12),
        child: Padding(
          padding: const EdgeInsets.all(12),
          child: Row(
            children: [
              // Checkbox
              Checkbox(
                value: task.completed,
                onChanged: (_) => onToggle(),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(4),
                ),
              ),

              const SizedBox(width: 12),

              // Conteúdo Principal
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
```

```
children: [
  // Título
  Text(
    task.title,
    style: TextStyle(
      fontSize: 16,
      fontWeight: FontWeight.bold,
      decoration: task.completed
          ? TextDecoration.lineThrough
          : null,
      color: task.completed
          ? Colors.grey
          : Colors.black,
    ),
    maxLines: 2,
    overflow: TextOverflow.ellipsis,
  ),

  if (task.description.isNotEmpty) ...[
    const SizedBox(height: 4),
    Text(
      task.description,
      style: TextStyle(
        fontSize: 14,
        color: task.completed
            ? Colors.grey.shade400
            : Colors.grey.shade700,
        decoration: task.completed
            ? TextDecoration.lineThrough
            : null,
      ),
      maxLines: 2,
      overflow: TextOverflow.ellipsis,
    ),
  ],

  const SizedBox(height: 8),

  // Metadata Row
  Row(
    children: [
      // Prioridade
      Container(
        padding: const EdgeInsets.symmetric(
          horizontal: 8,
          vertical: 4,
        ),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(12),
          border: Border.all(
            color: _getPriorityColor(),
            width: 1,
          ),
        ),
        child: Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            Icon(
              _getPriorityIcon(),
              size: 14,
              color: _getPriorityColor(),
            ),
            const SizedBox(width: 4),
            Text(
              _getPriorityLabel(),
              style: TextStyle(
                fontSize: 12,
                color: _getPriorityColor(),
                fontWeight: FontWeight.bold,
              ),
            ),
```

```
                        ],
                      ),
                    ),

                    const SizedBox(width: 12),

                    // Data
                    Icon(
                      Icons.access_time,
                      size: 14,
                      color: Colors.grey.shade600,
                    ),
                    const SizedBox(width: 4),
                    Text(
                      dateFormat.format(task.createdAt),
                      style: TextStyle(
                        fontSize: 12,
                        color: Colors.grey.shade600,
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),

          const SizedBox(width: 8),

          // Botão Deletar
          IconButton(
            onPressed: onDelete,
            icon: const Icon(Icons.delete_outline),
            color: Colors.red,
            tooltip: 'Deletar tarefa',
          ),
        ],
      ),
    ),
  ),
);
  }
}
```

PASSO 4: Atualizar Tela Principal

4.1 Atualizar `lib/screens/task_list_screen.dart`

```
import 'package:flutter/material.dart';
import '../models/task.dart';
import '../services/database_service.dart';
import '../widgets/task_card.dart';
import 'task_form_screen.dart';

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

  @override
  State<TaskListScreen> createState() => _TaskListScreenState();
}

class _TaskListScreenState extends State<TaskListScreen> {
  List<Task> _tasks = [];
  String _filter = 'all'; // all, completed, pending
  bool _isLoading = false;

  @override
```

```
void initState() {
  super.initState();
  _loadTasks();
}

Future<void> _loadTasks() async {
  setState(() => _isLoading = true);
  final tasks = await DatabaseService.instance.readAll();
  setState(() {
    _tasks = tasks;
    _isLoading = false;
  });
}

List<Task> get _filteredTasks {
  switch (_filter) {
    case 'completed':
      return _tasks.where((t) => t.completed).toList();
    case 'pending':
      return _tasks.where((t) => !t.completed).toList();
    default:
      return _tasks;
  }
}

Future<void> _toggleTask(Task task) async {
  final updated = task.copyWith(completed: !task.completed);
  await DatabaseService.instance.update(updated);
  await _loadTasks();
}

Future<void> _deleteTask(Task task) async {
  // Confirmar exclusão
  final confirmed = await showDialog<bool>(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('Confirmar exclusão'),
      content: Text('Deseja realmente excluir "${task.title}"?'),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context, false),
          child: const Text('Cancelar'),
        ),
        TextButton(
          onPressed: () => Navigator.pop(context, true),
          style: TextButton.styleFrom(foregroundColor: Colors.red),
          child: const Text('Excluir'),
        ),
      ],
    ),
  );

  if (confirmed == true) {
    await DatabaseService.instance.delete(task.id);
    await _loadTasks();

    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Tarefa excluída'),
          duration: Duration(seconds: 2),
        ),
      );
    }
  }
}

Future<void> _openTaskForm([Task? task]) async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(
```

```
      builder: (context) => TaskFormScreen(task: task),
    ),
  );

  if (result == true) {
    await _loadTasks();
  }
}

@override
Widget build(BuildContext context) {
  final filteredTasks = _filteredTasks;
  final stats = _calculateStats();

  return Scaffold(
    appBar: AppBar(
      title: const Text('Minhas Tarefas'),
      backgroundColor: Colors.blue,
      foregroundColor: Colors.white,
      elevation: 2,
      actions: [
        // Filtro
        PopupMenuButton<String>(
          icon: const Icon(Icons.filter_list),
          onSelected: (value) => setState(() => _filter = value),
          itemBuilder: (context) => [
            const PopupMenuItem(
              value: 'all',
              child: Row(
                children: [
                  Icon(Icons.list),
                  SizedBox(width: 8),
                  Text('Todas'),
                ],
              ),
            ),
            const PopupMenuItem(
              value: 'pending',
              child: Row(
                children: [
                  Icon(Icons.pending_actions),
                  SizedBox(width: 8),
                  Text('Pendentes'),
                ],
              ),
            ),
            const PopupMenuItem(
              value: 'completed',
              child: Row(
                children: [
                  Icon(Icons.check_circle),
                  SizedBox(width: 8),
                  Text('Concluídas'),
                ],
              ),
            ),
          ],
        ),
      ],
    ),

    body: Column(
      children: [
        // Card de Estatísticas
        if (_tasks.isNotEmpty)
          Container(
            margin: const EdgeInsets.all(16),
            padding: const EdgeInsets.all(16),
            decoration: BoxDecoration(
              gradient: const LinearGradient(
                colors: [Colors.blue, Colors.blueAccent],
```

```
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
              ),
              borderRadius: BorderRadius.circular(12),
              boxShadow: [
                BoxShadow(
                  blurRadius: 8,
                  offset: const Offset(0, 4),
                ),
              ],
            ),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceAround,
              children: [
                _buildStatItem(
                  Icons.list,
                  'Total',
                  stats['total'].toString(),
                ),
                _buildStatItem(
                  Icons.pending_actions,
                  'Pendentes',
                  stats['pending'].toString(),
                ),
                _buildStatItem(
                  Icons.check_circle,
                  'Concluídas',
                  stats['completed'].toString(),
                ),
              ],
            ),
          ),

        // Lista de Tarefas
        Expanded(
          child: _isLoading
              ? const Center(child: CircularProgressIndicator())
              : filteredTasks.isEmpty
                  ? _buildEmptyState()
                  : RefreshIndicator(
                      onRefresh: _loadTasks,
                      child: ListView.builder(
                        padding: const EdgeInsets.only(bottom: 80),
                        itemCount: filteredTasks.length,
                        itemBuilder: (context, index) {
                          final task = filteredTasks[index];
                          return TaskCard(
                            task: task,
                            onTap: () => _openTaskForm(task),
                            onToggle: () => _toggleTask(task),
                            onDelete: () => _deleteTask(task),
                          );
                        },
                      ),
                    ),
        ),
      ],
    ),

    floatingActionButton: FloatingActionButton.extended(
      onPressed: () => _openTaskForm(),
      icon: const Icon(Icons.add),
      label: const Text('Nova Tarefa'),
      backgroundColor: Colors.blue,
      foregroundColor: Colors.white,
    ),
  );
}

Widget _buildStatItem(IconData icon, String label, String value) {
  return Column(
```

```
    mainAxisSize: MainAxisSize.min,
    children: [
      Icon(icon, color: Colors.white, size: 32),
      const SizedBox(height: 4),
      Text(
        value,
        style: const TextStyle(
          color: Colors.white,
          fontSize: 24,
          fontWeight: FontWeight.bold,
        ),
      ),
      Text(
        label,
        style: const TextStyle(
          color: Colors.white70,
          fontSize: 12,
        ),
      ),
    ],
  );
}

Widget _buildEmptyState() {
  String message;
  IconData icon;

  switch (_filter) {
    case 'completed':
      message = 'Nenhuma tarefa concluída ainda';
      icon = Icons.check_circle_outline;
      break;
    case 'pending':
      message = 'Nenhuma tarefa pendente';
      icon = Icons.pending_actions;
      break;
    default:
      message = 'Nenhuma tarefa cadastrada';
      icon = Icons.task_alt;
  }

  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(icon, size: 100, color: Colors.grey.shade300),
        const SizedBox(height: 16),
        Text(
          message,
          style: TextStyle(
            fontSize: 18,
            color: Colors.grey.shade600,
          ),
        ),
        const SizedBox(height: 8),
        TextButton.icon(
          onPressed: () => _openTaskForm(),
          icon: const Icon(Icons.add),
          label: const Text('Criar primeira tarefa'),
        ),
      ],
    ),
  );
}

Map<String, int> _calculateStats() {
  return {
    'total': _tasks.length,
    'completed': _tasks.where((t) => t.completed).length,
    'pending': _tasks.where((t) => !t.completed).length,
  };
```

```
  }
}
```

PASSO 5: Atualizar Main

5.1 Atualizar `lib/main.dart`

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Task Manager Pro',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: Colors.blue,
          brightness: Brightness.light,
        ),
        useMaterial3: true,
        cardTheme: const CardThemeData(  // ← CardThemeData ao invés de CardTheme
          elevation: 2,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.all(Radius.circular(12)),
          ),
        ),
        inputDecorationTheme: const InputDecorationTheme(
          border: OutlineInputBorder(
            borderRadius: BorderRadius.all(Radius.circular(8)),
          ),
          filled: true,
          fillColor: Color(0xFFF5F5F5), // Colors.grey.shade50
        ),
      ),
      home: const TaskListScreen(),
    );
  }
}
```

PASSO 6: Criar Pasta Widgets

```
mkdir lib/widgets
```

Certifique-se que `task_card.dart` está em `lib/widgets/`.

PASSO 7: Executar e Testar

7.1 Executar Aplicação

```
flutter run
```

7.2 Cenários de Teste

Teste 1: Criar Nova Tarefa

1. Clique no FAB "Nova Tarefa"
2. Preencha título, descrição e prioridade
3. Clique em "Criar Tarefa"
4. Verifique aparição na lista

Teste 2: Editar Tarefa

1. Toque em um card de tarefa
2. Modifique os campos
3. Clique em "Atualizar Tarefa"
4. Verifique alterações

Teste 3: Validação

1. Tente criar tarefa sem título
2. Verifique mensagem de erro
3. Digite título com menos de 3 caracteres
4. Verifique validação

Teste 4: Filtros

1. Crie tarefas com diferentes status
2. Use o menu de filtros (ícone filtro)
3. Teste "Todas", "Pendentes", "Concluídas"
4. Verifique que apenas tarefas corretas aparecem

Teste 5: Marcar como Completa

1. Clique no checkbox de uma tarefa
2. Veja mudança visual (riscado, cinza)
3. Verifique estatísticas atualizadas
4. Desmarque e veja reversão

Teste 6: Deletar Tarefa

1. Clique no ícone de lixeira
2. Veja dialog de confirmação
3. Confirme exclusão
4. Verifique SnackBar de feedback

Teste 7: Prioridades

1. Crie tarefas com todas prioridades

2. Verifique cores diferentes nos cards:
   - Verde (Baixa)
   - Laranja (Média)
   - Vermelho (Alta)
   - Roxo (Urgente)

Teste 8: Pull-to-Refresh

1. Na lista, arraste para baixo
2. Solte para recarregar
3. Veja indicador de loading
4. Lista é atualizada

Teste 9: Estados Vazios

1. Delete todas as tarefas
2. Veja mensagem "Nenhuma tarefa"
3. Aplique filtro "Concluídas" sem ter nenhuma
4. Veja mensagem específica

Teste 10: Navegação

1. Entre no formulário
2. Clique "Cancelar"
3. Volte para lista sem salvar
4. Confirme que não criou tarefa

🎨 Customizações Opcionais

Customização 1: Temas Escuro/Claro

Adicione suporte a tema escuro no `main.dart`:

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Task Manager Pro',
      debugShowCheckedModeBanner: false,

      // Tema Claro
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: Colors.blue,
          brightness: Brightness.light,
        ),
        useMaterial3: true,
      ),

      // Tema Escuro
```

```
      darkTheme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: Colors.blue,
          brightness: Brightness.dark,
        ),
        useMaterial3: true,
      ),

      // Seguir configuração do sistema
      themeMode: ThemeMode.system,

      home: const TaskListScreen(),
    );
  }
}
```

Customização 2: Busca de Tarefas

Adicione barra de busca na `task_list_screen.dart`:

```
class _TaskListScreenState extends State<TaskListScreen> {
  // ... código existente
  String _searchQuery = '';

  List<Task> get _filteredTasks {
    var tasks = _tasks;

    // Filtro por status
    switch (_filter) {
      case 'completed':
        tasks = tasks.where((t) => t.completed).toList();
        break;
      case 'pending':
        tasks = tasks.where((t) => !t.completed).toList();
        break;
    }

    // Filtro por busca
    if (_searchQuery.isNotEmpty) {
      tasks = tasks.where((t) {
        return t.title.toLowerCase().contains(_searchQuery.toLowerCase()) ||
               t.description.toLowerCase().contains(_searchQuery.toLowerCase());
      }).toList();
    }

    return tasks;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Minhas Tarefas'),
        // ... resto do código
      ),
      body: Column(
        children: [
          // Barra de Busca
          Padding(
            padding: const EdgeInsets.all(16),
            child: TextField(
              decoration: InputDecoration(
                hintText: 'Buscar tarefas...',
                prefixIcon: const Icon(Icons.search),
                suffixIcon: _searchQuery.isNotEmpty
                    ? IconButton(
                        icon: const Icon(Icons.clear),
                        onPressed: () {
```

```
                        setState(() => _searchQuery = '');
                      },
                    )
                  : null,
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(12),
              ),
            ),
            onChanged: (value) {
              setState(() => _searchQuery = value);
            },
          ),
        ),

        // ... resto do código
      ],
    ),
  );
 }
}
```

Customização 3: Ordenação

Adicione opções de ordenação:

```
class _TaskListScreenState extends State<TaskListScreen> {
  // ... código existente
  String _sortBy = 'date'; // date, priority, title

  List<Task> get _filteredTasks {
    var tasks = /* ... filtros existentes ... */;

    // Ordenação
    switch (_sortBy) {
      case 'priority':
        final priorityOrder = {'urgent': 0, 'high': 1, 'medium': 2, 'low': 3};
        tasks.sort((a, b) {
          final orderA = priorityOrder[a.priority] ?? 2;
          final orderB = priorityOrder[b.priority] ?? 2;
          return orderA.compareTo(orderB);
        });
        break;
      case 'title':
        tasks.sort((a, b) => a.title.compareTo(b.title));
        break;
      case 'date':
      default:
        tasks.sort((a, b) => b.createdAt.compareTo(a.createdAt));
    }

    return tasks;
  }

  // No AppBar, adicione menu de ordenação
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        // ... código existente
        actions: [
          // Menu de Ordenação
          PopupMenuButton<String>(
            icon: const Icon(Icons.sort),
            onSelected: (value) => setState(() => _sortBy = value),
            itemBuilder: (context) => [
              const PopupMenuItem(
                value: 'date',
                child: Text('Ordenar por Data'),
              ),
```

```
              const PopupMenuItem(
                value: 'priority',
                child: Text('Ordenar por Prioridade'),
              ),
              const PopupMenuItem(
                value: 'title',
                child: Text('Ordenar por Título'),
              ),
            ],
          ),
          // ... resto dos actions
        ],
      ),
    );
  }
}
```

🐛 Troubleshooting

Erro: "Cannot find package 'widgets'"

Solução:

```
# Verifique estrutura de pastas
ls -la lib/widgets/

# Deve conter task_card.dart
# Se não existir, crie a pasta:
mkdir -p lib/widgets
```

Erro: "The method 'copyWith' isn't defined"

Solução: Certifique-se de que o método `copyWith` está implementado em `lib/models/task.dart`:

```
Task copyWith({
  String? title,
  String? description,
  bool? completed,
  String? priority,
}) {
  return Task(
    id: id,
    title: title ?? this.title,
    description: description ?? this.description,
    completed: completed ?? this.completed,
    priority: priority ?? this.priority,
    createdAt: createdAt,
  );
}
```

Erro: "setState() called after dispose()"

Solução: Sempre verifique `mounted` antes de chamar `setState`:

```
if (mounted) {
  setState(() {
    // seu código
  });
```

```
}
```

Cards não aparecem com estilo

Solução: Verifique que importou corretamente:

```
import '../widgets/task_card.dart'; // Caminho correto
```

Validação não funciona

Solução: Certifique-se de ter `GlobalKey<FormState>` e está chamando `validate()`:

```
final _formKey = GlobalKey<FormState>();

// No Form widget:
Form(
  key: _formKey,
  child: Column(/* ... */)
)

// Ao salvar:
if (_formKey.currentState!.validate()) {
  // Prosseguir com salvamento
}
```

📊 Comparação: Antes vs Depois

Aspecto	Laboratório 1	Laboratório 2
Telas	1 (Lista)	2 (Lista + Formulário)
Navegação	Nenhuma	Push/Pop entre telas
Componentes	Básicos	Material Design 3
Formulário	Inline simples	Completo com validação
Cards	ListTile simples	Cards customizados
UX	Básica	Profissional
Feedback	Nenhum	SnackBars, Dialogs
Filtros	Não	Sim (3 opções)
Estatísticas	Não	Sim (card gradiente)
Animações	Não	Transitions

✅ Checklist de Verificação

Funcionalidades Implementadas

- [ ] Tela de formulário separada
- [ ] Navegação entre telas funcionando

- [ ] Criar nova tarefa via formulário
- [ ] Editar tarefa existente
- [ ] Validação de campos obrigatórios
- [ ] Dropdown de prioridade
- [ ] Switch de status completo
- [ ] Cards customizados com visual profissional
- [ ] Ícones e cores por prioridade
- [ ] Botão de deletar com confirmação
- [ ] Filtros (todas/pendentes/concluídas)
- [ ] Card de estatísticas
- [ ] Estados vazios personalizados
- [ ] Pull-to-refresh
- [ ] SnackBars de feedback
- [ ] Dialogs de confirmação

Visual e UX

- [ ] Interface segue Material Design 3
- [ ] Cores consistentes
- [ ] Espaçamentos adequados
- [ ] Área de toque mínima 48x48
- [ ] Feedback visual em todas ações
- [ ] Loading states
- [ ] Empty states
- [ ] Animações suaves

Código

- [ ] Arquitetura em camadas mantida
- [ ] Widgets separados em arquivos
- [ ] Código comentado
- [ ] Sem warnings no console
- [ ] Tratamento de erros

📝 Entregável da Aula 2

🎯 Exercícios Complementares - Selecione pelo menos 2 (dois) para sua entrega

Exercício 1: Data de Vencimento

Adicione campo de data de vencimento:

- Adicione `DateTime? dueDate` ao modelo Task
- Use `DatePicker` no formulário
- Mostre alerta para tarefas vencidas
- Ordene por data de vencimento

Exercício 2: Categorias

Implemente sistema de categorias:

- Crie modelo Category
- Dropdown de categorias no formulário
- Filtro por categoria
- Cores diferentes por categoria

Exercício 3: Notificações Locais

Adicione lembretes:

- Instale `flutter_local_notifications`
- Campo de hora do lembrete
- Agende notificação
- Cancele ao completar tarefa

Exercício 4: Compartilhamento

Permita compartilhar tarefas:

- Instale `share_plus`
- Botão de compartilhar no card
- Formate texto para compartilhar
- Compartilhe via apps do sistema

Exercício 5: Backup/Restore

Implemente exportação de dados:

- Botão de exportar para JSON
- Salve arquivo no dispositivo
- Botão de importar JSON
- Validação dos dados importados

1. Código Fonte Completo

```
task_manager/
├── lib/
│   ├── main.dart (atualizado)
│   ├── models/
```

```
│   │   └── task.dart (do Lab 1)
│   ├── services/
│   │   └── database_service.dart (do Lab 1)
│   ├── screens/
│   │   ├── task_list_screen.dart (atualizado)
│   │   └── task_form_screen.dart (NOVO)
│   └── widgets/
│       └── task_card.dart (NOVO)
└── pubspec.yaml
```

2. Demonstração em Vídeo (2-3 minutos)

Grave mostrando:

1. **Tela inicial** - Lista com estatísticas
2. **Criar tarefa** - Abrir formulário, preencher, salvar
3. **Validação** - Tentar salvar sem título
4. **Editar tarefa** - Tocar em card, modificar, salvar
5. **Prioridades** - Mostrar diferentes cores
6. **Filtros** - Alternar entre todos/pendentes/concluídas
7. **Deletar** - Confirmar exclusão
8. **Pull-to-refresh** - Recarregar lista
9. **Estado vazio** - Mostrar quando não há tarefas

3. Screenshots

Tire prints de:

- Tela principal com tarefas
- Card de estatísticas
- Formulário de nova tarefa
- Formulário de edição
- Dialog de confirmação
- SnackBar de feedback
- Estado vazio
- Menu de filtros

4. Relatório Técnico (1 página)

Estrutura sugerida:

```
# Relatório - Laboratório 2: Interface Profissional

## 1. Implementações Realizadas
- Descreva as principais funcionalidades implementadas
- Liste os componentes Material Design 3 utilizados

## 2. Desafios Encontrados
- Quais dificuldades teve?
- Como resolveu?

## 3. Melhorias Implementadas
- O que adicionou além do roteiro?
```

```
- Customizações feitas

## 4. Aprendizados
- Principais conceitos aprendidos
- Diferenças entre Lab 1 e Lab 2

## 5. Próximos Passos
- O que gostaria de adicionar?
- Ideias para melhorar a aplicação
```

📚 Recursos Adicionais

Documentação Oficial

- **Material Design 3 Flutter** (https://m3.material.io/develop/flutter)
- **Navigation Cookbook** (https://docs.flutter.dev/cookbook/navigation)
- **Form Validation** (https://docs.flutter.dev/cookbook/forms/validation)
- **Gestures** (https://docs.flutter.dev/cookbook/gestures)

Packages Úteis

- `intl`: Formatação de datas
- `flutter_slidable`: Ações de deslizar
- `animations`: Transições animadas
- `flutter_staggered_animations`: Animações em lista

Tutoriais Recomendados

- **Material Design in Flutter** (https://www.youtube.com/watch?v=DL0Ix1lnC4w)

(https://www.youtube.com/watch?v=DL0Ix1lnC4w)

- **Navigation & Routing** (https://www.youtube.com/watch?v=nyvwx7o277U)

(https://www.youtube.com/watch?v=nyvwx7o277U)

- **Forms & Validation** (https://www.youtube.com/watch?v=S37yxR2O-FM)

(https://www.youtube.com/watch?v=S37yxR2O-FM)

💡 Dicas Finais

Para uma Interface de Qualidade

1. **Consistência é Chave**
   - Use mesmas cores em todo app
   - Mantenha espaçamentos uniformes
   - Padronize tamanhos de fonte
2. **Feedback Visual**
   - Sempre mostre loading
   - Confirme ações destrutivas
   - Use SnackBars para feedback rápido
3. **Acessibilidade**
   - Contraste adequado de cores
   - Tamanho mínimo de toque
   - Labels descritivas
4. **Performance**
   - Evite reconstruções desnecessárias
   - Use const constructors quando possível
   - Implemente lazy loading para listas grandes
5. **Testes de Usabilidade**
   - Teste em diferentes tamanhos de tela
   - Peça feedback de outros usuários
   - Verifique fluxo completo

🎓 Conceitos Avançados (Opcional)

State Management Alternativo

Para apps maiores, considere:

- **Provider**: Gerenciamento de estado simples
- **Bloc**: Padrão mais robusto
- **Riverpod**: Evolução do Provider
- **GetX**: Solução completa

Arquitetura Escalável

Para projetos reais:

- **Clean Architecture**: Separação em camadas
- **MVVM**: Model-View-ViewModel
- **Repository Pattern**: Abstração de dados
- **Dependency Injection**: Desacoplamento

Desenvolvido para: PUC Minas - Engenharia de Software
Disciplina: Laboratório de Desenvolvimento de Aplicações Móveis e Distribuídas
Versão: 2.0 - 2025