NOMES: Lucas Schmidt - Luiz Henrique

1. Descrição do Projeto

Tema escolhido

Lista de Tarefas Gamificada

Objetivo do aplicativo

O aplicativo tem como objetivo auxiliar os usuários a organizarem suas tarefas diárias de forma divertida, utilizando elementos de gamificação como pontos, níveis e conquistas. A proposta é aumentar o engajamento e a produtividade dos usuários ao transformar suas metas em desafios recompensadores.

Funcionalidades implementadas

- Login com nome do usuário
- Criação e exclusão de tarefas
- Marcação de tarefas como concluídas
- Sistema de conquistas baseado na quantidade de tarefas concluídas
- Exibição de estatísticas com barra de progresso
- Interface simples com navegação por abas (BottomNavigationBar)

2. Estrutura do Projeto

Organização em pastas

```
lib/
├── main.dart
├── models/
│   ├── tarefa.dart
│   └── usuario.dart
├── pages/
│   ├── login_page.dart
│   ├── home_page.dart
│   ├── tarefas_page.dart
│   ├── conquistas_page.dart
│   ├── estatisticas_page.dart
└── widgets/
    ├── tarefa_card.dart
    └── progresso_bar.dart
```

Frameworks e bibliotecas utilizadas

- **Flutter** (framework principal)
- **Material Design** (interface padrão do Flutter)
- (Planejado) **shared_preferences** ou **Hive** para persistência local

Explicação sobre o fluxo de telas e navegação

1. O usuário acessa a LoginPage e insere seu nome.
2. Após login, o app navega para a HomePage, que possui três abas:

- **Tarefas**: lista de tarefas com opção de adicionar e marcar como concluída.
- **Conquistas**: exibe o progresso e níveis atingidos.
- **Estatísticas**: mostra número total de tarefas e progresso em porcentagem.

3. Aplicação de Padrões de Projeto

Padrões Criacionais: Singleton

Problema: Garantir que o controle de usuário (nome, pontos, nível) seja único durante toda a sessão.
Aplicação: (Planejado) A classe Usuario pode ser gerenciada como um singleton para centralizar o estado do usuário.

Padrões Estruturais: Adapter

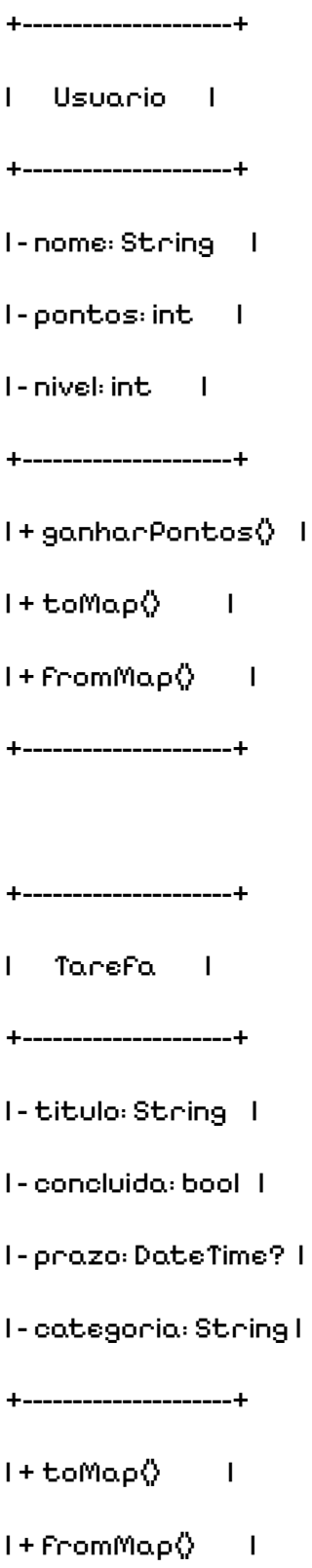
Problema: Adaptar o modelo Tarefa para formatos compatíveis com armazenamento (JSON/Map).
Aplicação: O padrão Adapter é utilizado nos métodos toMap() e fromMap() nas classes Tarefa e Usuario, permitindo integração com bancos locais ou APIs.

Padrões Comportamentais: Observer

Problema: Atualizar a interface sempre que o estado das tarefas muda.
Aplicação: A classe TarefasPage usa setState() para refletir mudanças imediatamente na interface — simulando o comportamento do padrão Observer. Com Provider ou Riverpod, isso poderia ser melhor implementado.

4. Diagrama do Projeto

Diagrama de Classes (UML)



+--------------------+

Relacionamentos:

HomePage --> TarefasPage, ConquistasPage, EstatisticasPage

TarefasPage --> List<Tarefa>

ConquistasPage --> int (tarefas concluídas)

(Recomenda-se criar esse diagrama visualmente com Draw.io, Lucidchart, ou outra ferramenta UML.)

8. Conclusão

O que foi aprendido com o projeto

- Organização de projetos Flutter em camadas (models, pages, widgets)
- Manipulação de estado com setState e navegação com BottomNavigationBar
- Uso de gamificação para aumentar o engajamento do usuário
- Implementação de padrões de projeto básicos

Dificuldades encontradas

- Modularização do código e separação de responsabilidades
- Atualização automática da interface com múltiplos widgets e estados
- Gerenciamento da persistência de dados ainda em fase de planejamento

Sugestões de melhorias futuras

- Adicionar persistência de dados com shared_preferences ou Hive
- Implementar Provider para gerenciar o estado globalmente
- Permitir edição e exclusão de tarefas
- Criar sistema de notificações para tarefas com prazo
- Integração com Firebase para autenticação e armazenamento em nuvem
- Criar conquistas e níveis dinâmicos com base em XP e categorias