

# Programação Para Dispositivos Móveis

Aluno:  
Alexandre Candido

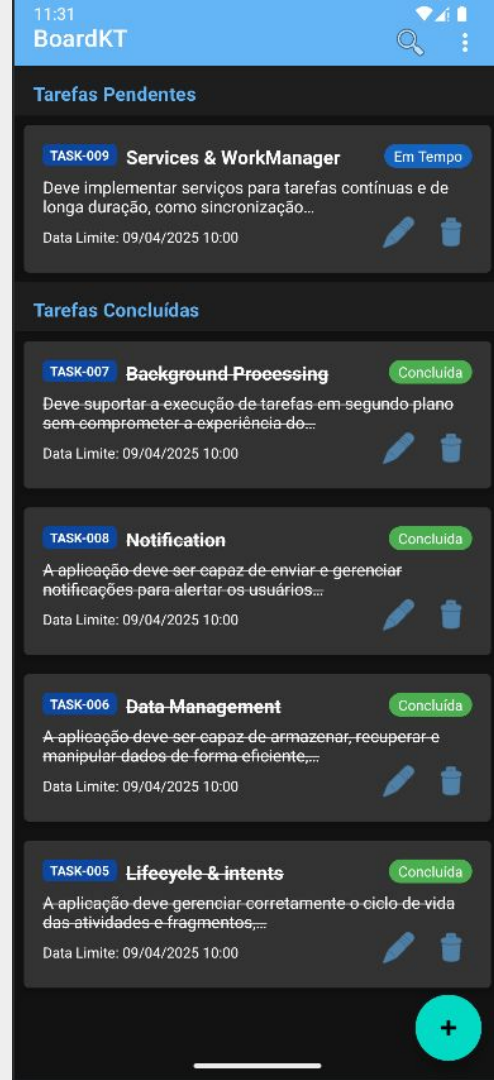
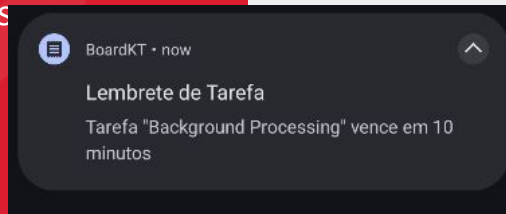


UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

# BoardKT

## Aplicação de gerenciamento de tarefas

- Fácil criação e edição de tarefas a serem feitas.
- Notificações quando as tarefas estão próximas de atingir a data limite imposta
- Busca e ordenação facilitadas
- Dados salvos em banco de dados local mas podem ser exportados e importados em outros dispositivos com facilidade.
- Funciona de forma responsiva em diferentes telas.



# Requisitos

## Gestão de Lifecycle:

- MainActivity: utiliza onCreate() para inicializar componentes e observa a LiveData do Room, que gerencia automaticamente questões de ciclo de vida.
- AddTaskActivity: preserva o estado durante edição/criação de tarefas e gerenciar navegação de retorno.

## Preservação de Estado:

- Dados das tarefas são persistidos no banco de dados Room, garantindo que não haja perda de informações.

## Uso de Intents:

- Navegação explícita entre atividades usando Intent.
- Transferência de dados entre atividades usando extras (putExtra/getExtra)

# Lifecycle & Intents

# Data Management

## Arquitetura de Persistência:

- Utiliza o *Room Database* para abstração do *SQLite*
- Implementa o padrão *DAO* (Data Access Object)

## Queries Eficientes:

- **Ordenação:** *getAllTasksByDeadline()* e *getAllTasksByCreation()*
- **Filtros:** queries *SQL* para separar tarefas concluídas e pendentes
- **Busca:** *searchTasks()* com parâmetros *LIKE* para pesquisa parcial

## Integração com Componentes do Sistema:

- *LiveData* para atualização automática de UI quando dados mudam
- *Coroutines* para operações assíncronas evitando bloqueio da thread principal
- *Singleton* para instância do banco de dados que sobrevive ao ciclo de v

# Background Processing

## Corrotinas para operações assíncronas:

- Utiliza *lifecycleScope.launch* para operações de banco de dados sem bloquear a thread principal
- Implementa funções *suspend* para operações assíncronas no DAO (como *insertTask*, *updateTask*)
- Escopo de corrotina apropriado (*Dispatchers.IO*) para operações intensivas de I/O

## WorkManager para Tarefas Agendadas:

- Implementa um *Worker* para execução em segundo plano
- Usa *OneTimeWorkRequest* para agendar notificações
- Persiste tarefas mesmo após reinicialização do dispositivo

## LiveData para Atualizações Reativas:

- Observa resultados de banco de dados com LiveData para atualização automática da UI
- Elimina manipulação manual de threads ao vincular dados do banco à interface

# Services, WorkManager & Notifications

## WorkManager para Notificações Programadas:

- Utiliza TaskReminderWorker para agendar e executar notificações de lembretes
- Garante a entrega confiável de notificações mesmo após reinicialização do dispositivo

## Reescalonamento Inteligente:

- Função *scheduleAllReminders()* para restaurar todos os lembretes ao iniciar o aplicativo
- Evita agendamentos duplicados com *cancelAllWorkByTag()* antes de criar novos trabalhos
- Verifica condições antes de agendar (tarefas concluídas, prazos já passados)
- Implementa mecanismo de cancelamento e reagendamento quando tarefas são modificadas

## Operações Assíncronas em Segundo Plano:

### Otimização de Banco de Dados:

- Aplica índices implícitos em colunas de chave primária (Task.id)
- Define queries específicas para cada caso de uso (filtragem, ordenação, busca)

### Carregamento Eficiente da UI:

- Implementa RecyclerView para reutilização eficiente de views
- Utiliza ViewHolder para cache de referências de views e reduzir usos de findViewById()
- Usa LiveData para atualizar a UI apenas quando os dados mudam

### Gerenciamento de Recursos:

- Singleton do banco de dados para evitar múltiplas instâncias.
- Liberação adequada de recursos quando tarefas são concluídas ou excluídas

# Performance



# Usability

## Implementação do Material Design:

- Utiliza componentes como *MaterialCardView*, *FloatingActionButton*, *Toolbar* e *SearchView*

## Organização Visual Eficiente:

- Separa tarefas em seções claras ("Tarefas Pendentes" e "Tarefas Concluídas")
- Utiliza *RecyclerView* com layout adaptável para diversas densidades de tela
- Aplica cores primárias e secundárias consistentes através do aplicativo

## Feedback ao Usuário:

- Fornece notificações para lembretes de tarefas
- Exibe diálogos de confirmação para ações de exclusão

## Acessibilidade:

- Define `contentDescription` para buttons iconográficos
- Possui suporte para tema escuro
- Mantém contraste adequado entre texto e fundo

## Proteção Contra Injeção SQL:

- Utiliza Room como ORM sobre o SQLite, que parametriza automaticamente consultas SQL
- Evita concatenação direta de strings em consultas SQL com `@Query` parametrizada

## Armazenamento Seguro de Dados:

- Usa o armazenamento interno do aplicativo que é privado por padrão (sandbox)
- Sistema de arquivos Android isolado por usuário e aplicativo

## Boas Práticas de Permissões:

- Solicita permissões de notificação explicitamente
- Verifica permissões em tempo de execução antes de exibir notificações

## Validação de Entrada:

- Implementa validação de campos em formulários antes de salvar no banco de dados

# Security

# Compatibility

## Suporte a Múltiplas Versões do Android:

- Implementa verificação condicional de versão
- Oferece tratamento específico para permissões em Android 13+
- Implementa *NotificationChannel* apenas para Android 8.0+

## Layout Responsivo e Flexível:

- Aplica *ConstraintLayout* como base dos layouts, oferecendo posicionamento relativo eficiente
- Define *ScrollView* em formulários para garantir acessibilidade em telas pequenas
- Implementa margens e paddings baseados em dp (não px) para escala adequada

## Gerenciamento de Recursos:

- Externaliza strings para localização em *strings.xml*
- Define dimensões em unidades relativas (dp, sp) em vez de absolutas
- Implementa *maxLines* e *ellipsize* para lidar com textos longos
- Usa *wrap\_content* e *match\_parent* apropriadamente para elementos flexíveis.