Estruturação e Planejamento do Projeto Final

Desenvolvimento de Aplicativos Móveis

📋 Informações do Projeto

- Equipe:
  - Pedro Miguel Radwanski
  - John Claude Cameron Chappell
  - Gustavo Rossi
  - Felipe Vieira
- Data de Entrega: 30/10/2025
- Tema do Aplicativo: Pokédex com Montagem de Time

1️⃣ Etapa 1 — Definição do Tema e Funcionalidades Principais

- 🎯 Tema Central
  - [Um aplicativo Pokédex que permite aos usuários navegar por uma lista de Pokémon obtidos da PokeAPI, ver seus detalhes e, após autenticação com Firebase, montar um time pessoal de até 6 Pokémon.]
- ⚙️ Requisitos Funcionais:

ID	Descrição da Funcionalidade
RF01	O usuário deve poder visualizar uma lista paginada de Pokémon.
RF02	O usuário deve poder tocar em um Pokémon da lista para ver seus detalhes (nome, tipo, habilidades, estatísticas, sprite).
RF03	O usuário deve poder se cadastrar e logar com email e senha.
RF04	O usuário logado deve poder adicionar um Pokémon ao seu time (limite de 6).

RF05	O usuário logado deve poder visualizar seu time atual.
RF06	O usuário logado deve poder remover um Pokémon do seu time.

- 🌐 Integração de Rede
  Abordagem escolhida: [X] Retrofit e [X] Firebase
  (Nota: Ambas são necessárias. Retrofit para a PokeAPI pública e Firebase para dados do usuário/time).

Operações de rede (Retrofit - PokeAPI):

- GET - Buscar lista paginada de Pokémon (`/pokemon`).
- GET - Buscar detalhes de um Pokémon específico (`/pokemon/{id_ou_nome}`).
- Operações de rede (Firebase - Times):
- GET - Buscar o time do usuário logado (Firestore).
- POST - Adicionar um Pokémon ao time (Firestore).
- DELETE - Remover um Pokémon do time (Firestore).

API/Serviço utilizado:

- https://www.google.com/search?q=https://pokeapi.co/api/v2/ (Via Retrofit)
- Firebase Authentication (Para login)
- Firebase Firestore (Para salvar os times dos usuários)
- 📱 Esboço Rápido das Telas
  - Tela de [Login/Cadastro]
    - Função: [Autenticar (login) ou registrar (cadastro) o usuário com email e senha.]
    - Operações: [Remota (Firebase Authentication)]
  - Tela de [Pokédex (Lista)]
    - Função: [Exibir a lista principal de Pokémon. Deve suportar paginação.]
    - Operações: [Remota (Retrofit p/ PokeAPI) + Local (Cache com Room)]

- Tela de [Detalhes do Pokémon]
  - Função: [Exibir informações detalhadas (tipos, stats, sprite) de um Pokémon selecionado. Inclui o botão "Adicionar ao Time" se logado.]
  - Operações: [Remota (Retrofit p/ PokeAPI) + Local (Cache com Room)]
- Tela de [Meu Time]
- Função: [Exibir os 6 Pokémon selecionados pelo usuário logado.]
- Operações: [Remota (Firestore) + Local (Cache/Espelho com Room)]

2️⃣ Etapa 2 — Fluxo de Navegação

🗺️ Diagrama de Navegação

[Tela de Login/Cadastro]
↓ (Se logado com sucesso)
[Tela Principal (Lista Pokédex)] ↔ (Aba/Menu) ↔ [Tela "Meu Time"]
↓ (Clique em um item)
[Tela de Detalhes do Pokémon]
↓ (Clique "Adicionar ao Time")
(Salva no Firebase)

📍 Descrição do Fluxo
Sequência de navegação:

1. O usuário abre o aplicativo e é apresentado à Tela de Login/Cadastro.
2. Após a autenticação bem-sucedida, ele é direcionado para a Tela Principal (Lista Pokédex).
3. Na lista, ele pode rolar (carregando mais Pokémon via paginação) e clicar em qualquer Pokémon.
4. Ao clicar, navega para a Tela de Detalhes, onde vê as informações e o sprite do Pokémon.
5. Na Tela de Detalhes, um botão "Adicionar ao Time" está visível. Ao clicar, o Pokémon é salvo no time do usuário no Firebase (se houver menos de 6).
6. O usuário pode, a qualquer momento (via Bottom Navigation ou Menu), navegar para a Tela "Meu Time" para ver sua equipe atual e remover

membros.

🔄 Tipo de Requisição por Tela

- TelaTipo de OperaçãoFonte de Dados[Tela Login/Cadastro]Requisição remotaFirebase Auth[Tela Pokédex (Lista)]Leitura remota + CacheRetrofit (PokeAPI) + Room[Tela Detalhes]Leitura remota + CacheRetrofit (PokeAPI) + Room[Tela Meu Time]Leitura/Escrita remota + SyncFirebase Firestore + Room
- 🏗️ Estratégia de Navegação
  Abordagem escolhida:
  - Múltiplas Activities (navegação tradicional com Intents)
    [X] Activity única + Navigation Compose

    Justificativa:
    [Navigation Compose é a abordagem idiomática para UI declarativa. Facilita o gerenciamento de estado entre telas, a passagem de argumentos (como o ID do Pokémon) e lida de forma robusta com o fluxo condicional de navegação (usuário logado vs. não logado) em uma única Activity.]

3️⃣ Etapa 3 — Planejamento do Banco de Dados (Room)

🗄️ Entidades (Tabelas)
(Nota: Room será usado como cache para a PokeAPI e como "fonte única de verdade" local para o time do usuário, que é espelhado do Firestore.)

Entidade 1: [PokemonCacheEntity] (Cache de detalhes)

```
@Entity(tableName = "pokemon_cache")
data class PokemonCacheEntity(
    @PrimaryKey
    val id: Int, // ID vindo da PokeAPI
    val name: String,
    val imageUrl: String,
    val type1: String,
```

```
    val type2: String?, // Pokémon pode ter só um tipo
    val height: Int,
    val weight: Int
    // Outros atributos (stats) podem ser armazenados como JSON string ou co
lunas
)
```

Atributos:

id (Int) - Chave primária (ID da PokeAPI)
name (String) - Nome do Pokémon
imageUrl (String) - URL do sprite
type1/type2 (String) - Tipos do Pokémon

Entidade 2: [TeamMemberEntity] (Time do usuário logado)

```
@Entity(tableName = "user_team")
data class TeamMemberEntity(
    @PrimaryKey
    val pokemonId: Int, // ID da PokeAPI, garante que só pode adicionar um de
cada
    val userId: String, // FK para o usuário do Firebase
    val name: String, // Denormalizado para exibição rápida
    val imageUrl: String, // Denormalizado
    val slotPosition: Int // Posição de 1 a 6 (opcional, para ordenação)
)
```

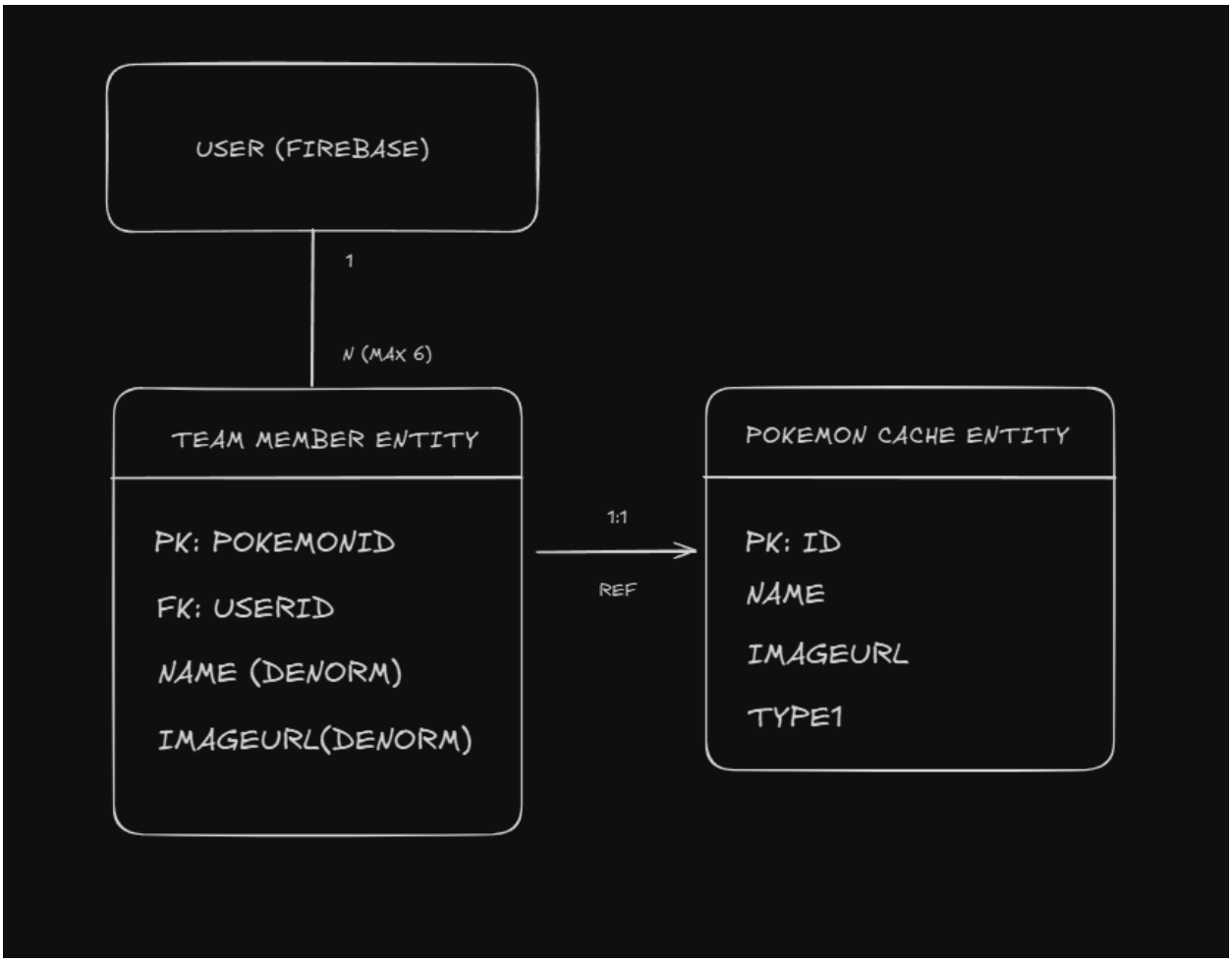
Atributos:

pokemonId (Int) - Chave primária (ID da PokeAPI)
userId (String) - ID do usuário do Firebase
name (String) - Nome do Pokémon
imageUrl (String) - URL do sprite

🔗 Relacionamentos entre Entidades
Entidade 1	Relacionamento	Entidade 2	Descrição
User (Firebase)	1:N	TeamMemberEntity	[Um usuário (identificado por `userId`)

possui até 6 `TeamMemberEntity`] |
| PokemonCacheEntity | 1:1 (Ref) | TeamMemberEntity | [Um `TeamMemberEntity` referencia os dados de um `PokemonCacheEntity` através do `pokemonId`] |

📊 Diagrama Entidade-Relacionamento



🔄 Estratégia de Sincronização
Persistência local:
[Room armazenará dados cacheados da PokeAPI (Entidade 1) para reduzir chamadas de rede. Também armazenará o time do usuário logado (Entidade 2) como um espelho do Firestore.]

Sincronização com backend:

- Sincronização automática em tempo real (Para o Time, via Firebase)
- Sincronização manual (pull-to-refresh)

- Sincronização em background
- Apenas dados locais (sem sincronização)

Fluxo:

1. **Time (Firestore → Room):** Ao logar, o app ativa um `snapshotListener` no Firestore (`/users/{userId}/team`). Qualquer mudança (adição/remoção) no Firestore é imediatamente salva na tabela `TeamMemberEntity` do Room. As UIs (Tela "Meu Time") observam *apenas* o Flow do Room (fonte única de verdade).
2. **Pokédex (PokeAPI → Room):** Ao carregar a lista ou detalhes, o Repository primeiro verifica o Room. Se os dados existirem e forem recentes (cache válido), retorna do Room. Se não, busca da PokeAPI (Retrofit), salva no `PokemonCacheEntity` e então retorna os dados.

4️⃣ Etapa 4 — Integração Remota

🌐 Abordagem de Rede Escolhida

Opção: [X] Retrofit e [X] Firebase
(Ambas são essenciais para os requisitos)

📡 Configuração Retrofit (se aplicável)

Base URL: `https://pokeapi.co/api/v2/`

Endpoints utilizados:

Método	Endpoint	Descrição	Parâmetros
GET	`/pokemon`	Buscar lista paginada	`offset: Int`, `limit: Int`
GET	`/pokemon/{id}`	Buscar item específico por ID	`id: Int`
GET	`/pokemon/{name}`	Buscar item específico por Nome	`name: String`

🔥 Configuração Firebase (se aplicável)

Serviços utilizados:

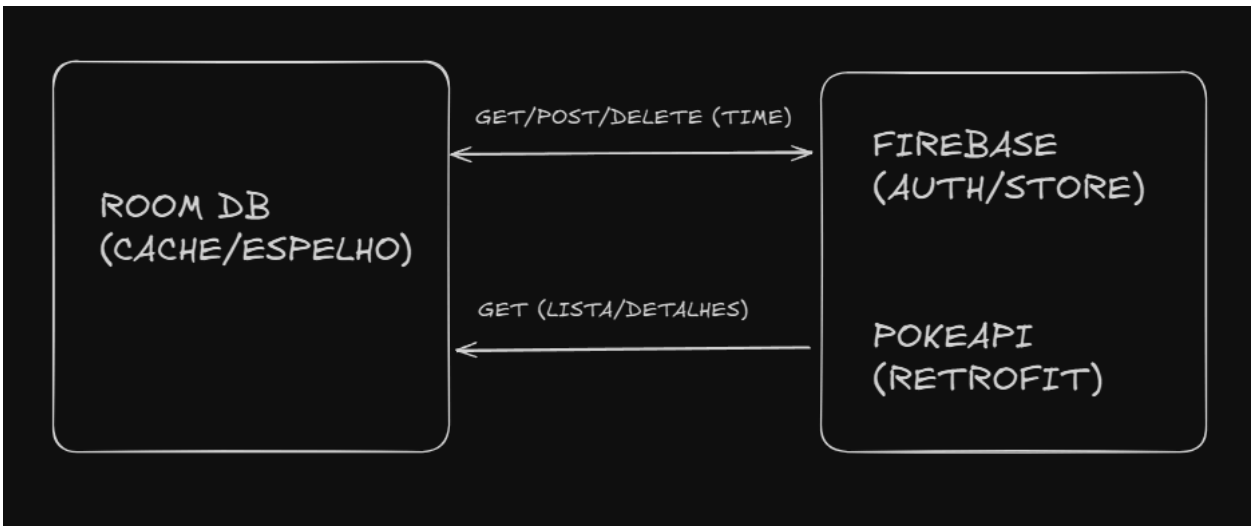
- [X] Firebase Authentication (Para login/cadastro com email e senha)
- [X] Firebase Firestore (Para salvar os times)
- ☐ Firebase Realtime Database

☐ Firebase Storage

Coleções/Estruturas:

```
/users
  ├─ {userId} (Documento do usuário)
  │   ├─ email: "usuario@email.com"
  │   │
  │   └── /team (Subcoleção do time)
  │       │
  │       ├─ {pokemonId_1} (Ex: "25")
  │       │   ├─ name: "pikachu"
  │       │   ├─ imageUrl: "url/do/sprite.png"
  │       │   └─ addedAt: Timestamp
  │       │
  │       ├─ {pokemonId_2} (Ex: "4")
  │       │   ├─ name: "charmander"
  │       │   ...
  │       │
  │       └─ (Até 6 documentos nesta subcoleção)
```

🔄 Diagrama de Sincronização



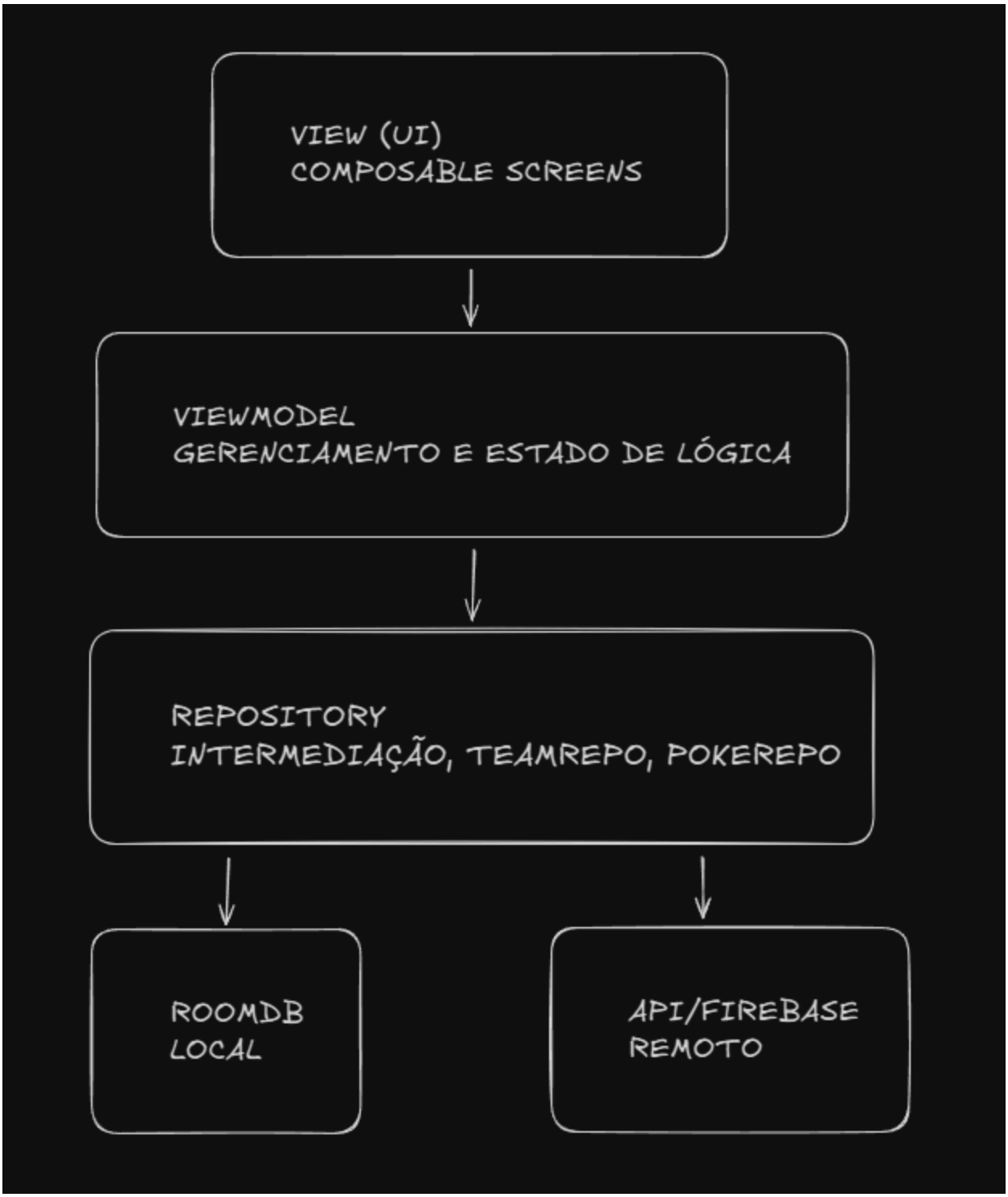
Estratégia:

1. **Time (Firebase ↔ Room):** A fonte da verdade (master) é o Firestore. O Room atua como um espelho offline/reativo. Escritas (Adicionar/Remover) vão para o Firestore. Leituras (UI) vêm do Room, que é atualizado pelo listener do Firestore.
2. **Pokédex (PokeAPI → Room):** A fonte da verdade é a PokeAPI. O Room é apenas um cache "read-through".
3. **Falhas de Rede (PokeAPI):** Se a PokeAPI falhar, a UI exibirá os dados do cache (Room), se disponíveis.
4. **Falhas de Rede (Firebase):** O SDK do Firebase gerencia o cache offline. As operações de escrita ficarão na fila e o listener local refletirá o estado offline, mantendo o Room e a UI consistentes.

5️⃣ Etapa 5 — Considerações Técnicas

🏛️ Arquitetura do Projeto

Padrão arquitetural: MVVM (Model-View-ViewModel)



📚 Bibliotecas e Dependências

- **Core Android:** Kotlin, Android SDK
- **Arquitetura e Ciclo de Vida:** ViewModel, Lifecycle Components

- **Interface de Usuário:** Jetpack Compose, Material Design 3, Navigation Compose
- **Persistência Local:** Room Database
- **Rede (PokeAPI):** Retrofit, OkHttp
- **Rede (Usuário/Time):** Firebase SDK (Authentication, Firestore)
- **Processamento Assíncrono:** Kotlin Coroutines, StateFlow (para o UiState)
- **Serialização:** Kotlinx.serialization (para Retrofit e PokeAPI DTOs)
- **Injeção de Dependências:** Hilt (para injetar ViewModels, Repositories, Daos, etc.)
- **Imagens:** Coil (para carregar os sprites dos Pokémon)

⚡ Uso de Corrotinas
Contextos utilizados:

- `Dispatchers.Main` - Atualizar o `_uiState` (StateFlow) no ViewModel.
- `Dispatchers.IO` - Chamadas de rede (Retrofit/Firestore) e acesso ao banco de dados (Room).
- `Dispatchers.Default` - (Não deve ser muito usado aqui, talvez para filtrar uma lista muito grande em memória).

Exemplo de implementação:

```
viewModelScope.launch {
    _uiState.value = UiState.Loading
    try {
        // Busca dados da PokeAPI ou Room via repositório
        val data = withContext(Dispatchers.IO) {
            // Ex: pokemonRepository.getPokemonDetails(pokemonId)
        }
        _uiState.value = UiState.Success(data)
    } catch (e: Exception) {
        _uiState.value = UiState.Error(e.message)
```

```
    }
}
```

🎨 Estrutura de Pacotes

```
com.example.pokedexapp/
├── data/
│   ├── local/ (Room)
│   │   ├── dao/ (PokemonDao, TeamDao)
│   │   ├── entities/ (PokemonCacheEntity, TeamMemberEntity)
│   │   └── PokedexDatabase.kt
│   ├── remote/
│   │   ├── firebase/ (AuthService, FirestoreService)
│   │   ├── pokeapi/
│   │   │   ├── dto/ (PokemonListDto, PokemonDetailDto)
│   │   │   └── PokeApiService.kt (Interface Retrofit)
│   └── repository/ (PokemonRepository, TeamRepository, UserRepository)
├── domain/ (Opcional, mas recomendado)
│   ├── model/ (Pokemon, TeamMember)
│   └── usecase/ (GetPokemonListUseCase, AddToTeamUseCase)
├── ui/ (Compose)
│   ├── screens/
│   │   ├── login/
│   │   ├── pokedex_list/
│   │   ├── pokemon_detail/
│   │   └── team/
│   ├── components/ (PokemonCard, TypeChip, StatBar)
│   └── theme/
├── di/ (Módulos do Hilt - AppModule, FirebaseModule, NetworkModule)
└── PokedexApplication.kt (Entry point HUnilt)
```

📝 Resumo Executivo

Principais Funcionalidades

- Visualização de lista paginada e detalhes de Pokémon (via PokeAPI).

- Autenticação de usuário (Cadastro/Login com email e senha via Firebase Auth).
- Montagem de time personalizado (máx. 6 Pokémon) salvo no Firebase Firestore.

Tecnologias Chave

- Kotlin, Jetpack Compose, MVVM
- Room (Cache local e espelho do time)
- Retrofit (Consumo da PokeAPI)
- Firebase Authentication & Firestore (Gerenciamento de usuário e time)
- Coroutines & Flow (Programação assíncrona e reativa)
- Hilt (Injeção de Dependência)

Diferencial do Projeto
[A aplicação combina o consumo de uma API REST pública (PokeAPI) com um backend de usuário (Firebase), utilizando Room como uma camada de cache robusta e como fonte única de verdade para a UI, demonstrando uma arquitetura moderna e reativa.]

Desafios Previstos

- Gerenciar duas fontes de dados remotas distintas (Retrofit e Firebase) em um único repositório ou coordenar múltiplos repositórios.
- Implementar a lógica de sincronização (Firestore → Room) de forma eficiente e sem conflitos.
- Lidar corretamente com a paginação (Paging 3) da PokeAPI integrada ao cache do Room.
- Gerenciar o estado de autenticação (logado/deslogado) e exibir/ocultar condicionalmente as funcionalidades de "Time".
  📅 Cronograma de Desenvolvimento (Opcional)
  EtapaDescriçãoPrazo1Setup inicial, Hilt, Firebase, Retrofit, Room[Data]2Implementação do Login/Cadastro (Firebase Auth) [Data]3Implementação da Pokédex (Retrofit + Room Cache) [Data]4Implementação do "Meu Time" (Firestore + Room Sync)

[Data]5Desenvolvimento das UIs (Compose) de todas as telas[Data]6Testes, polimento e entrega final[Data]
✅ Critérios de Avaliação

- Clareza na definição do tema e funcionalidades
- Organização do fluxo de navegação
- Modelagem adequada do banco de dados
- Planejamento da integração remota
- Demonstração de conhecimento técnico
- Viabilidade de implementação

Documento elaborado por: Pedro Miguel, John Claude, Felipe Vieira e Gustavo Rossi