Estruturação e Planejamento

Sistema de Gerenciamento de Reservas de Hotel

Lucca Alves de Souza Anderle	**37717651**
Matheus Machado da Mota	**37823779**

Sumário

1. Objetivo

O objetivo deste documento é descrever as principais funcionalidades de um sistema de gerenciamento de reservas de hotel construído utilizando a linguagem Kotlin.

1.1. Requisitos Funcionais

1.1.1. RF01 - O usuário poderá cadastrar quartos;
1.1.2. RF02 - O usuário poderá cadastrar hóspedes;
1.1.3. RF03 - O usuário poderá cadastrar reservas;
1.1.4. RF04 - O usuário poderá editar e excluir quartos;
1.1.5. RF05 - O usuário poderá editar e excluir hóspedes;
1.1.6. RF06 - O usuário poderá editar e excluir reservas;
1.1.7. RF07 - O aplicativo exibirá uma lista com todos os quartos;
1.1.8. RF08 - O aplicativo exibirá uma lista com todos os hóspedes;
1.1.9. RF09 - O aplicativo exibirá uma lista com todas as reservas;
1.1.10. RF10 - O usuário deve ser capaz de registrar o check-in, alterando o status do quarto para “ocupado”;
1.1.11. RF11 - O usuário deve ser capaz de registrar o check-out, alterando o status do quarto para “disponível”;
1.1.12. RF12 - O sistema deve ser capaz de impedir reservas de serem realizadas em quartos com reservas atreladas no mesmo período desejado;
1.1.13. RF13 - O usuário deve ser capaz de realizar login antes de poder realizar outras operações;
1.1.14. RF14 - O usuário deve ser capaz de realizar logoff.

1.2. Operações que Envolverão Rede

A persistência de dados do aplicativo será implementada em uma arquitetura híbrida para garantir funcionalidade offline e performance otimizada.

Firebase Firestore será utilizado para:

- **Sincronização em Nuvem:** Usado para manter os dados (Quartos, Hóspedes e Reservas) sincronizados em tempo real entre todos os dispositivos logados.
- **Backup:** Atua como o repositório central e seguro de todos os dados do sistema.
- **Autenticação:** O Firebase Authentication será o único responsável pelo Login e Logoff de usuários.

Room será utilizado para:

- **Persistência Local:** Atua como a "Fonte Única de Verdade" do aplicativo, armazenando cópias locais de todos os dados essenciais.
- **Funcionalidade Offline:** Permite que o usuário visualize, filtre e, em alguns casos, realize operações de escrita (que serão sincronizadas posteriormente) mesmo sem conexão de rede.

1.3. Esboço das Telas

1.3.1. Tela de Login

Tela inicial do aplicativo com a opção para fazer login. A tela contará com dois campos de input solicitando o nome de usuário e a senha e um botão para realizar o login. Assim que ocorrer a autenticação, o usuário é levado para a tela principal.

1.3.2. Tela Principal

A tela principal possui no canto superior direito um botão de logoff, que ao ser clicado, exibe uma confirmação para ter certeza de que o usuário quer mesmo realizar o logoff. Caso confirmado, ele realiza o logoff e retorna o usuário à tela inicial.

Mais abaixo, existe um menu de navegação que permite o usuário navegar entre as diferentes sessões do aplicativo, sendo essas:

- Quartos;
- Hóspedes;
- Reservas.

1.3.3. Tela de Quartos

Essa tela exibe uma lista com todos os quartos, incluindo seu número, tipo e status. O usuário pode clicar em cada quarto para abrir uma tela de edição. Embaixo da lista existe um botão que permite ao usuário adicionar um novo quarto caso quando interagido. A tela também possuí no canto superior esquerdo a opção de voltar para a tela principal e no canto superior direito a opção de realizar logoff.

1.3.3.1. Tela de Edição de Quartos

Mostra todos os dados do quarto selecionado em campos de input editáveis. Na parte inferior da tela existem os botões de “salvar” para salvar as alterações e retornar à tela anterior e o botão “excluir” para excluir o quarto. O botão de “excluir” pede uma confirmação antes de realizar a

ação e retornar o usuário à tela anterior. Na parte superior da tela, assim como nas outras telas, existe o botão de retornar à página anterior e o botão de realizar logoff.

1.3.3.2. Tela de Cadastro de Quartos

Essa tela segue o mesmo modelo da Tela de Edição de Quartos, com a diferença sendo que os campos de input aparecem vazios.

1.3.4. Tela de Hóspedes

Essa tela segue o mesmo padrão da Tela de Quartos. Uma lista mostrando todos os hóspedes, incluindo o nome, CPF, E-mail e telefone. Quando o usuário clica em um hóspede ele é levado para a tela de edição. Os botões para retornar à tela inicial e logoff estão presentes na parte superior e embaixo da lista existe o botão de cadastro de hóspedes.

1.3.4.1. Tela de Edição de Hóspedes

Segue o modelo apresentado na Tela de Edição de Quartos.

1.3.4.2. Tela de Cadastro de Hóspedes

Segue o modelo apresentado na Tela de Cadastro de Quartos.

1.3.5. Tela de Reservas Ativas

Essa tela segue o mesmo padrão da Tela de Quartos. Uma lista mostrando todas as reservas ativas, incluindo o hóspede associado à reserva, a data de check-in, check-out e o status. No topo da lista, existe um botão que leva o usuário que mostra uma lista de todas as reservas já concluídas. Quando o usuário clica em uma reserva ele é levado para a tela de edição. Os botões para retornar à tela inicial e logoff estão presentes na parte superior e embaixo da lista existe o botão de cadastro de reservas.

1.3.5.1. Tela de Reservas Concluídas

Essa tela tem os botões de voltar à tela anterior e logoff na parte superior e embaixo exibe uma lista com todas as reservas com o status “Concluída”.

1.3.5.2. Tela de Edição de Reservas

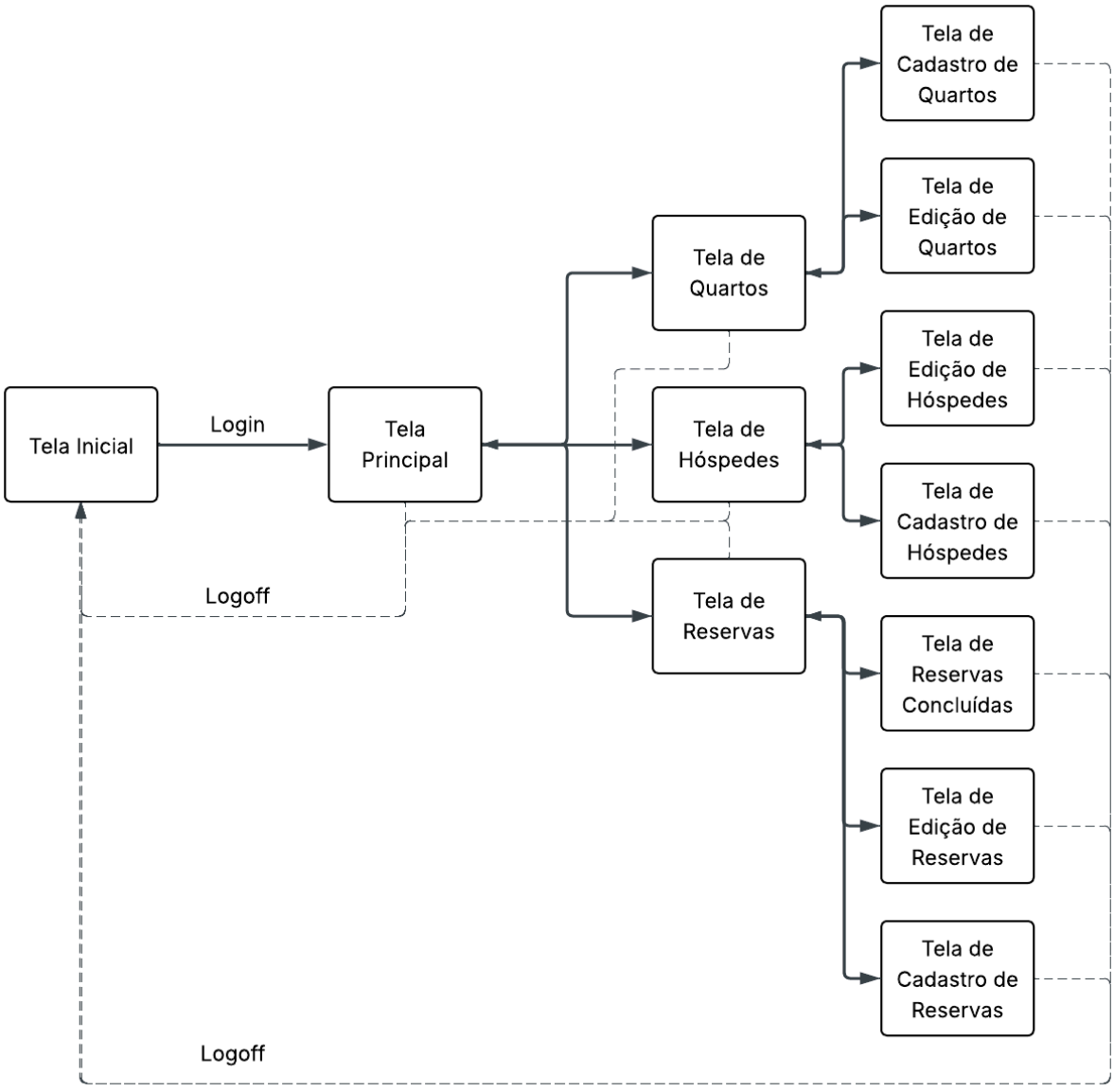
Segue o modelo apresentado na Tela de Edição de Quartos.

1.3.5.3. Tela de Cadastro de Reservas

Segue o modelo apresentado na Tela de Cadastro de Quartos.

2. Fluxo de Navegação

2.1. Diagrama de Fluxo



2.2. Fluxo de Usuário

2.2.1. Acesso Inicial:

O usuário começa na **Tela de Login**.

2.2.2. Autenticação:

Na **Tela de Login**, o usuário realiza a autenticação:

Se for bem-sucedida, ele é direcionado à **Tela Principal**.

Se falhar, ele permanece na **Tela de Login**.

2.2.3. Navegação Principal:

A **Tela Principal** serve como um painel de controle, onde o usuário pode acessar as três grandes áreas do sistema: **Quartos, Hóspedes e Reservas**.

Logoff: O botão de Logoff permite que o usuário retorne à **Tela de Login**.

2.2.4. Fluxo de Gestão (Quartos, Hóspedes):

A partir das listas (Quartos ou Hóspedes), o usuário pode iniciar um **Cadastro** ou selecionar um item para **Edição**.

Após **Salvar** ou **Excluir** nas telas de **Edição/Cadastro**, o usuário retorna à respectiva lista.

2.2.5. Fluxo de Reservas:

A partir da lista, o usuário pode iniciar um **Cadastro** ou selecionar um item para **Edição**.

Após **Salvar** ou **Excluir** nas telas de **Edição/Cadastro**, o usuário retorna à lista.

Existe um caminho adicional para a **Lista de Reservas Concluídas** (apenas para visualização).

2.3. Requisições e Formas de Navegação

O aplicativo se utiliza de uma única **Activity** com **Navigation Compose** para navegação entre as telas. As telas de lista (**Quartos**, **Hóspedes** e **Reservas**) utilizam **Room** para persistência, enquanto as telas de **Cadastro** e **Edição/Exclusão** utilizam tanto o **Room** para persistência, quanto o **Firebase** para a sincronização. A tela de **Login** e o **Logoff** se utilizam exclusivamente do **Firebase** para autenticação.

3. Planejamento do Banco de Dados

3.1. Lista das Entidades

3.1.1. usuario

Atributo	Tipo	Restrição
id	String	PRIMARY KEY
nomeUsuario	String	UNIQUE, NOT NULL
email	String	UNIQUE, NOT NULL
senha	String	NOT NULL

3.1.2. quarto

Atributo	Tipo	Restrição
id	String	PRIMARY KEY
numero	Int	UNIQUE, NOT NULL
tipo	String	NOT NULL
valorDiaria	Double	NOT NULL
status	String	NOT NULL

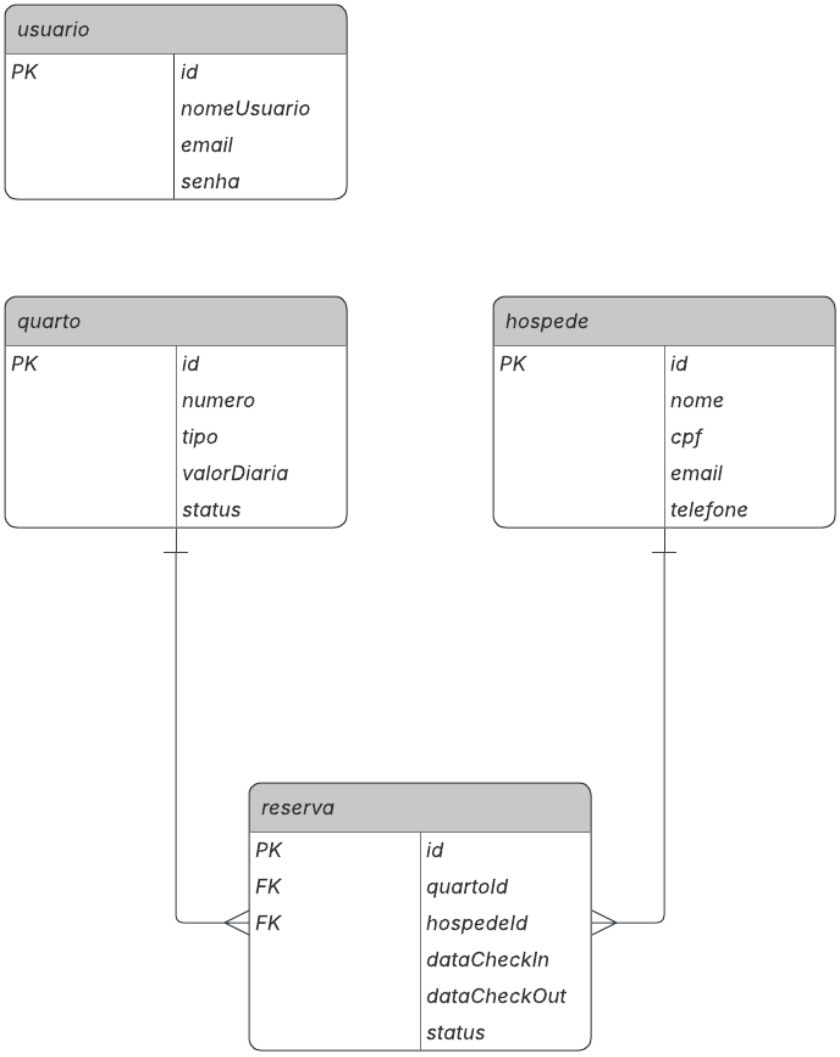
3.1.3. hospede

Atributo	Tipo	Restrição
id	String	PRIMARY KEY
nome	String	NOT NULL
cpf	String	UNIQUE, NOT NULL
email	String	
telefone	String	

3.1.4. reserva

Atributo	Tipo	Restrição
id	String	PRIMARY KEY
quartoId	String	FOREIGN KEY, NOT NULL
hospedeId	String	FOREIGN KEY, NOT NULL
dataCheckIn	Date	NOT NULL
dataCheckOut	Date	NOT NULL
status	String	NOT NULL

3.2. Diagrama de Banco de Dados



3.3. Persistência e Sincronização

Room é a camada de **persistência** local, responsável por armazenar todas as entidades e permitir consultas eficientes. A UI do aplicativo irá observar o **Room** (via Flow ou LiveData) para garantir que os dados exibidos sejam sempre os mais atualizados localmente.

O **Firebase Firestore** atua como a API de **sincronização** e backup:

- **Leitura:** O **Firebase** busca as últimas alterações da nuvem e as envia para o **Room**, mantendo o banco de dados local atualizado.

- **Escrita:** Qualquer operação de **Cadastro**, **Edição** ou **Exclusão** é primeiro salva no **Room** e, em seguida, propagada para o **Firebase Firestore** para garantir que todos os outros dispositivos sejam sincronizados.
- **Autenticação:** O **Firebase Authentication** é usado exclusivamente para gerenciar as sessões de **Login/Logoff**.

4. Integração de Rede

4.1. Firebase

A abordagem de integração de rede escolhida é o **Firebase**, utilizando os serviços **Firebase Authentication** para gestão de usuários e **Cloud Firestore** para persistência e sincronização de dados. Esta escolha suporta o requisito de uma arquitetura híbrida.

4.1.1. Estrutura

As coleções no **Firestore** serão mapeadas diretamente a partir das entidades **Room** definidas no planejamento. Cada documento terá um id único gerado pelo **Firestore**.

- **usuario (Coleção: users):** Gerenciado principalmente pelo **Firebase Authentication**, este documento espelho incluirá os campos nomeUsuario (string), email (string) e senha (string).
- **quarto (Coleção: rooms):** Incluirá numero (number), tipo (string), valorDiaria (number) e status (string, com valores "disponível" ou "ocupado").
- **hospede (Coleção: guests):** Possuirá os campos nome (string), cpf (string, usado como campo único), email (string, opcional) e telefone (string, opcional).
- **reserva (Coleção: reservations):** Conterá as referências quartoId (string) e hospedeId (string), além de dataCheckin (timestamp), dataCheckOut (timestamp) e status (string, com valores "ativa", "concluída" ou "cancelada").

4.1.2. Operações

As operações **CRUD** e de busca serão realizadas de forma assíncrona para garantir a sincronização em tempo real com o banco local (**Room**).

- **Buscar/Listar** (Requisição Conceptual: getDocs e onSnapshot): Utilizadas nas coleções **rooms**, **guests** e **reservations** para sincronizar os dados e fornecer as listas necessárias em tempo real. O uso do onSnapshot (Listener) é preferencial para manter a UI atualizada.

- **Cadastrar** (Requisição Conceptual: addDoc / setDoc): Usadas nas coleções **rooms**, **guests** e **reservations** para registrar novos quartos, hóspedes e reservas.
- **Editar** (Requisição Conceptual: updateDoc): Aplicadas em **rooms**, **guests** e **reservations** para modificar informações. Esta operação também é central para a gestão de **check-in/check-out**, atualizando o status em **rooms** e **reservations**.
- **Excluir** (Requisição Conceptual: deleteDoc): Utilizada em rooms, guests e reservations para remoção de dados.
- **Autenticação** (Requisição Conceptual: signInWithEmailAndPassword, signOut): Direcionadas ao **Firebase Auth** para gerenciamento de **Login** e **Logoff**.

4.1.3. Sincronização

O sistema adotará o padrão **Single Source of Truth (SSoT)**, onde o banco de dados local **Room** é o provedor primário de dados para a interface de usuário (UI). O **Firestore** atua como a camada de sincronização e backup em nuvem.

Fluxo de Sincronização (Firebase - Room - UI)

Este fluxo é fundamental para garantir a funcionalidade offline e o tempo real (real-time).

1. **Sincronização de Leitura (Cloud-to-Local)**
   - 1.1. **Observação do Cloud:** O aplicativo estabelece listeners de tempo real (onSnapshot) nas coleções do **Firestore** (**rooms, guests, reservations**).
   - 1.2. **Atualização do Cloud:** Quando um dado é alterado na nuvem (por outro dispositivo ou pelo próprio backend), o **Firestore** envia a atualização ao dispositivo.
   - 1.3. **Atualização do Room:** O repositório do aplicativo recebe os dados do **Firestore** e os armazena/atualiza no banco de dados **Room**.
2. **Sincronização de Escrita (Local-to-Cloud)**

2.1. **Operação Local:** Qualquer operação de Cadastro, Edição ou Exclusão (CRUD) é executada primeiro no banco de dados **Room**.

2.2. **Atualização Imediata da UI:** A UI é notificada e atualiza a exibição imediatamente, garantindo uma experiência de usuário fluida e responsiva.

2.3. **Propagação para o Cloud:** Após a conclusão da operação no **Room**, o repositório emite a mesma operação (e os dados atualizados) para o **Firestore**.

2.4. **Sincronização Final:** O **Firestore** aceita a alteração e, via seu mecanismo de tempo real, propaga essa alteração para todos os outros dispositivos logados, garantindo a consistência global.

5. Considerações Técnicas

5.1. Arquitetura

Para estruturar o sistema e garantir a funcionalidade *offline* com sincronização em tempo real, a arquitetura de software a ser adotada será o padrão **Model-View-ViewModel (MVVM)**.

O **MVVM** suportará o princípio do **Single Source of Truth (SSOT)**, onde o banco de dados local **Room** será o provedor primário de dados para a interface do usuário, enquanto o **Firebase** atuará como a camada de sincronização e backup. As operações de leitura e escrita serão orquestradas para manter a UI atualizada de forma reativa e garantir a consistência global dos dados.

5.2. Bibliotecas

O projeto utilizará as seguintes bibliotecas:

- Room
- Firebase Firestore
- Firebase Authentication
- ViewModel / LiveData / KotlinFlow
- Jetpack Compose
- Navigation Compose

5.3. Corrotinas

O uso de corrotinas se dará nas seguintes situações:

- **Operações de Banco de Dados:** Todas as operações **CRUD** no **Room** serão executadas de forma assíncrona usando **Corrotinas** para evitar o bloqueio da linha de execução principal (UI Thread).
- **Sincronização de Rede:** As requisições de escrita (Cadastro, Edição, Exclusão) para o **Firebase Firestore**, bem como as chamadas de autenticação, serão tratadas em um contexto de **Corrotina**.
- **Fluxo de Sincronização:** As **Corrotinas** permitem implementar o fluxo de sincronização de forma eficiente: a operação **CRUD** é feita primeiro no **Room**, e em seguida, propagada para o **Firebase Firestore** para sincronização em nuvem (Fluxo **Local-to-Cloud**).