PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

Escola politécnica

Análise e desenvolvimento de sistema

Victor Hugo Aguiar Porfiro,

Josué Félix da Silva,

Guilherme Basto Borge,

Kevin Oliveira Matos,

Pedro Henrique Talalayv Gomes.

Projeto Integrador

Sistema de gerenciamento de filas hospitalares

2024 Goiânia, Go

1. Introdução

- A inspiração para este trabalho nasceu ao presenciar uma cena em um hospital público de Aparecida de Goiânia: uma gestante, com um quadro de sangramento, aguardava horas por um exame. A necessidade básica de se alimentar a fez ausentar-se por um momento, o que lhe custou a vaga. Esse desencontro, que expôs mãe e bebê a um risco prolongado, motivou a reflexão que se segue.

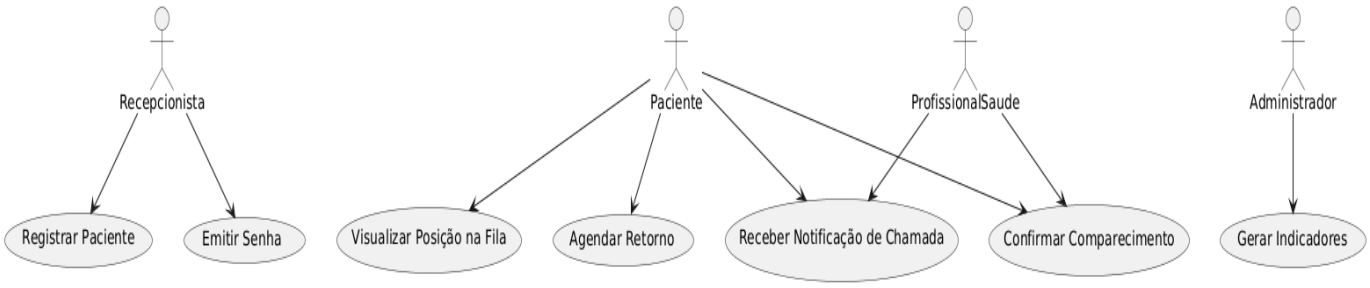
  Visando mitigar ocorrências como essa, este trabalho propõe o desenvolvimento de um sistema de gerenciamento de filas hospitalares. Por meio dele, o paciente cadastrado e esperando na fila do hospital vai ter ciência de quantos outros pacientes estão à sua frente, uma estimativa de minutos para o seu atendimento e receberá uma notificação quando for efetivamente chamado para a consulta ou exame e assim nasce o SGFH que promete acabar com essas ocorrências e garantir que todos tenham acesso as necessidades básicas.

2. Definição do escopo

- Projeto Model Canva (PMC):

3. Especificação de requisitos

- Requisitos funcionais (RF):
  - Cadastro de Pacientes: O sistema deve permitir o cadastro dos dados pessoais e de contato dos pacientes.
  - Gerenciamento de Cadastros: Possibilitar que o hospital gerencie esses cadastros para futuras notificações.
  - Notificações em Tempo Real: Enviar notificações via aplicativo informando, por exemplo, a chamada para atendimento/exame médico.
  - Exibição de Status: Mostrar o status do atendimento e o tempo estimado de espera.
  - Restrições de Agendamento: Impedir marcação de novos atendimentos se houver pendências ou chamadas não atendidas.
  - Registro de Interações: Registrar todas as notificações e interações para controle e auditoria.
  - Lembretes Automáticos: Enviar lembretes dos compromissos agendados.
  - Coleta de Feedback: Permitir que os pacientes insiram feedback sobre o atendimento recebido.
  - Integração com Sistemas Hospitalares: Sincronizar dados com outras plataformas e sistemas internos do hospital.
- Requisitos não funcionais (RNF):
  - Interface Intuitiva: Uma interface amigável que facilita o uso pelos pacientes e funcionários.
  - Segurança e Confiabilidade: Proteção dos dados e garantia de funcionamento estável.
  - Compatibilidade Multi-plataforma: O sistema deve funcionar bem em diferentes dispositivos e ambientes.
  - Escalabilidade (parte de Integração e Escalabilidade): Permitir a expansão e a conexão com outros sistemas conforme a demanda.
- Diagrama de casos de uso:



4. Arquitetura de software

- Ferramentas e tecnologias utilizadas:

  - Java: É uma linguagem de programação orientada a objetos, conhecida por sua portabilidade. Amplamente utilizada no desenvolvimento de aplicações robustas e de grande escala, desde sistemas corporativos até aplicativos móveis e foi escolhida por sua versatilidade.

  - Spring Boot: É um framework que simplifica o desenvolvimento de aplicações Java e nos ajudou na integração da nossa aplicação com o banco de dados PostgreSQL. Ele facilita a configuração, o desenvolvimento e a implantação de aplicações baseadas em Spring.

  - Firebase: É uma plataforma de desenvolvimento de aplicativos móveis e web do Google. Oferece uma variedade de ferramentas e serviços, como banco de dados NoSQL em tempo real (Firestore/Realtime Database), autenticação, hosting, armazenamento de arquivos e notificações push, agilizando o desenvolvimento do backend.

  - PostgreSQL: É um sistema de gerenciamento de banco de dados relacional (SGBDR) de código aberto, conhecido por sua robustez, extensibilidade e conformidade com os padrões SQL. É uma escolha popular para aplicações que exigem confiabilidade e integridade de dados.

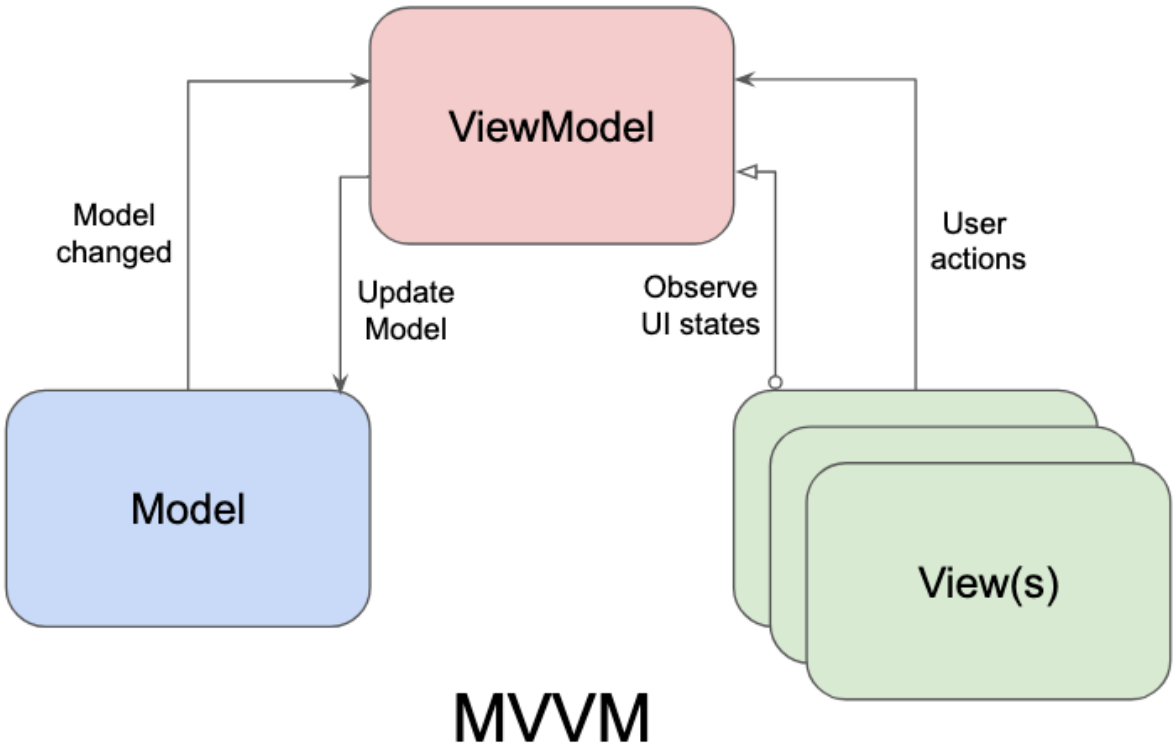
  - Swagger (OpenAPI): É uma especificação e um conjunto de ferramentas para descrever, projetar, construir e documentar APIs RESTful. Permite que tanto humanos quanto máquinas entendam as capacidades de um serviço web sem acesso ao código-fonte, facilitando a integração e o consumo de APIs.

  - Postman: É uma plataforma colaborativa para desenvolvimento de APIs. É amplamente utilizado para testar, documentar e compartilhar APIs, permitindo que os desenvolvedores enviem requisições HTTP para os endpoints de uma API e visualizem as respostas.

  - Gradle: É uma ferramenta de automação de compilação de código aberto, focada em flexibilidade e performance. É comumente usada em projetos Java (incluindo Android e Spring Boot) para gerenciar dependências, compilar código, executar testes e empacotar a aplicação para distribuição.

- Arquiteturas:

  Para garantir clareza na responsabilidade de cada componente, alta testabilidade e facilidade de manutenção, adotei dois padrões de arquitetura complementares no nosso sistema. No front-end Android, optei pelo MVVM (ModelView-ViewModel). A View (Activities e Fragments) fica responsável apenas por renderizar dados e capturar eventos de interação; o ViewModel expõe estados através de LiveData ou StateFlow, orquestra fluxos de dados assíncronos com Kotlin Coroutines e delega operações ao repositório; o Model, por sua vez, é organizado em Repositories (que abstraem fontes de dados como Retrofit e Room) e DataSources (que implementam a comunicação com APIs REST e armazenamento local). Com essa separação, o código torna-se modular, facilitando testes unitários isolados de cada camada, e a injeção de dependências via Hilt/Dagger assegura baixo acoplamento e maior flexibilidade na evolução do projeto



No back-end, construído em Spring Boot, adotei uma arquitetura em camadas clássica:

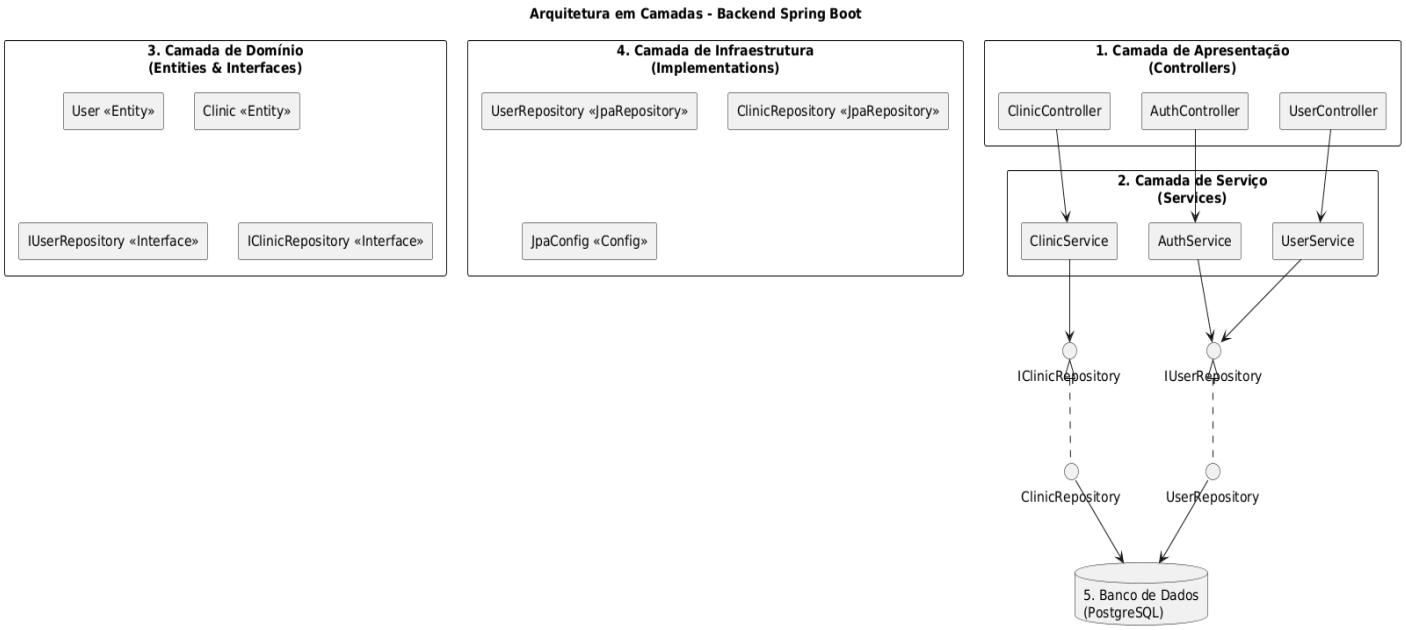
1. Controller: expõe endpoints REST, valida entradas (usando anotações de validação do Bean Validation) e converte entidades em DTOs para comunicação com o cliente.

2. Service: concentra a lógica de negócio — regras de fila de atendimento, agendamento de retornos e envio de notificações — e gerencia transações através de @Transactional.

3. Repository: abstrai o acesso a dados com Spring Data JPA (ou integração a MongoDB), permitindo operações CRUD de forma declarativa.

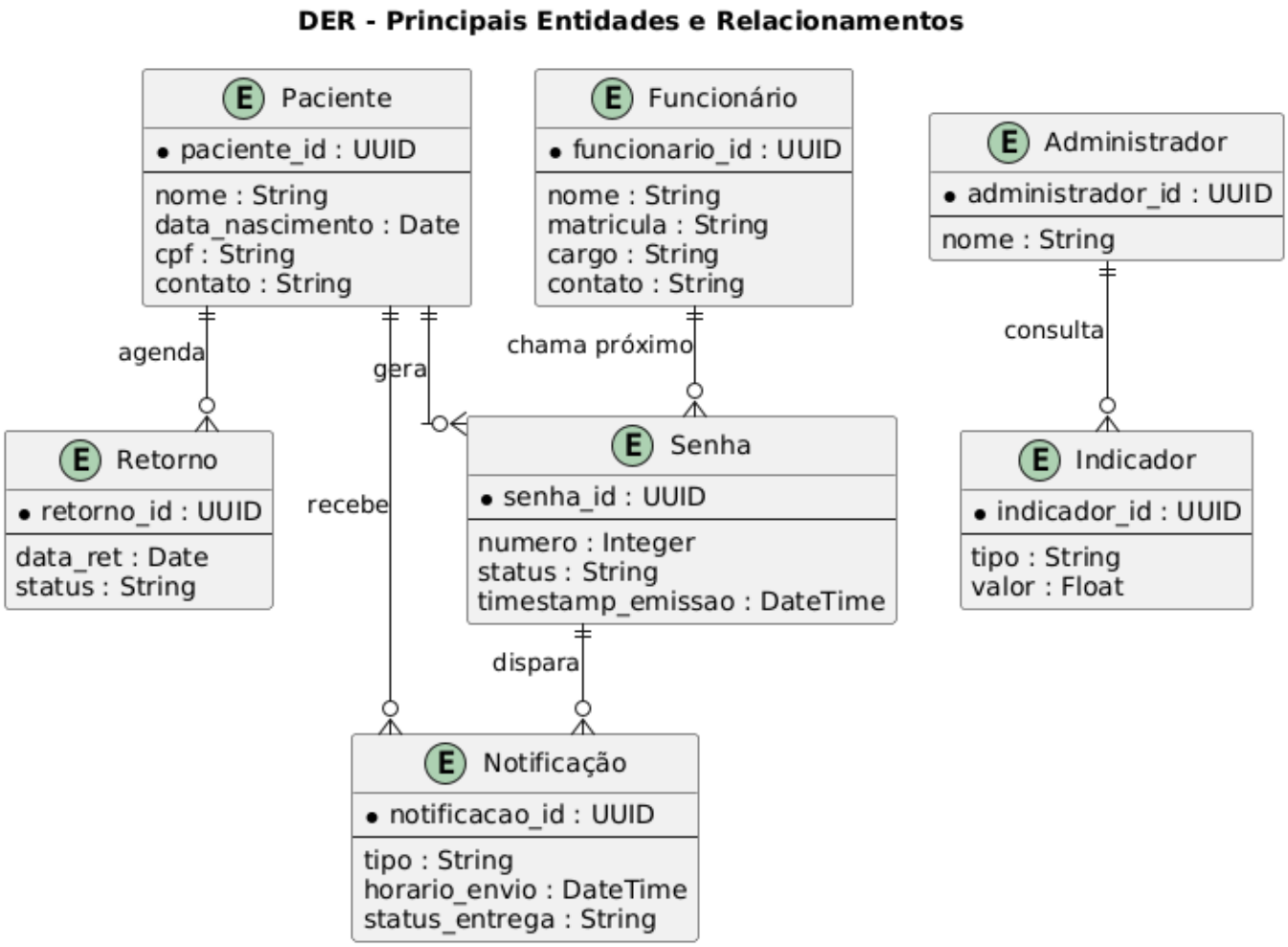
4. DTOs: objetos de transferência desacoplam a representação interna (entidades) da interface JSON, evitando vazamento de detalhes de persistência.

Essa divisão assegura que mudanças na regra de negócio não impactem diretamente a camada de apresentação, e facilita a adoção de testes de unidade e de integração em cada nível, mantendo coesão interna e reduzindo a complexidade do código. Em conjunto, o MVVM no Android e a arquitetura em camadas no Spring Boot promovem uma base sólida e escalável: cada alteração ou nova funcionalidade pode ser isolada em sua camada correspondente, reduzindo riscos de regressão e permitindo que equipes diferentes (front-end e back-end) trabalhem de modo independente, mas sempre alinhados aos mesmos princípios de projeto.



5. Projeto do banco de dados

- Diagrama Entidade-Relacionamento (DER):



- Descrição das tabelas, chaves e relacionamentos:

➢ Tabela “Paciente”

A entidade **Paciente** armazena dados pessoais dos pacientes atendidos pela clínica. Sua chave primária é o atributo paciente_id (UUID), que identifica unicamente cada registro. Além disso, essa tabela contém os seguintes atributos: nome (String), que guarda o nome completo do paciente; data_nascimento (Date), que registra a data de nascimento; cpf (String), que armazena o CPF (único por paciente) e serve como campo de validação de unicidade; e contato (String), que registra telefone ou e-mail para eventual comunicação.

Internamente, a tabela **Retorno** (descrita adiante) carrega a foreign key paciente_id apontando para esta tabela, indicando que um mesmo paciente pode agendar vários retornos ao longo do tempo. Além disso, a tabela **Senha** e a tabela **Notificação** também referenciam paciente_id para ligar a cada chamado ou notificação ao paciente correspondente. Assim, o relacionamento “um para muitos” se dá da forma:

- Um único paciente (1) → pode ter zero ou muitos (0..*) retornos agendados.

- Um único paciente (1) → pode gerar zero ou muitas senhas de atendimento.

- Um único paciente (1) → pode receber zero ou muitas notificações.

➢ Tabela “Funcionário”

A entidade **Funcionário** guarda informações dos profissionais de saúde (médicos, enfermeiros, recepcionistas, etc.) que utilizam o sistema para chamar pacientes. A chave primária dessa tabela é funcionario_id (UUID), que identifica cada profissional de forma única. Os demais atributos são: nome (String), matricula (String) — código interno ou matrícula do profissional —, cargo (String), informando a função (por exemplo, “Enfermeiro” ou “Psicólogo”), e contato (String), que pode ser telefone ou e-mail.

A tabela **Senha** possui uma coluna funcionario_id como foreign key, fazendo referência ao registro da tabela **Funcionário**. Dessa forma, cada vez que um profissional “chama o próximo paciente”, uma nova senha de atendimento é registrada, apontando para o funcionario_id responsável. O relacionamento exato é:

- Um único funcionário (1) → pode chamar zero ou muitas senhas (0..*) ao longo do expediente.

➢ Tabela “Administrador”

A entidade **Administrador** representa o usuário com perfil de administrador do sistema, cujo papel principal é consultar indicadores gerenciais. Seu identificador único é administrador_id (UUID), e ela possui apenas mais um atributo: nome (String). Como nem todos os administradores necessariamente geram indicadores, a quantidade de registros em **Indicador** pode ser zero para determinado administrador, mas todo **Indicador** sempre referência um administrador específico. Logo, o relacionamento é:

- Um único administrador (1) → pode possuir zero ou muitos indicadores (0..*).

➢ Tabela “Retorno”

A tabela **Retorno** armazena os agendamentos de retorno que os profissionais liberam para os pacientes. Sua chave primária é retorno_id (UUID). São registrados também o atributo data_ret (Date), que indica a data prevista para o retorno, e status (String), que informa se o retorno ainda está ativo, cancelado ou concluído. Além disso, cada retorno carrega a foreign key paciente_id, que aponta para a tabela **Paciente**. Assim, o vínculo exato é:

- Cada retorno (muitos) pertence a um único paciente (1).

- Cada paciente (1) pode ter zero ou muitos retornos (0..*).

➢ Tabela “Senha”

A entidade **Senha** representa um chamado para atendimento em fila. A chave primária é senha_id (UUID). Os atributos adicionais são número (Integer), que corresponde ao número sequencial atribuído à senha; status (String), que pode assumir valores como “Aguardando”, “Chamada” ou “Finalizada”; e timestamp_emissao (DateTime), que registra o momento em que a senha foi gerada ou chamada. Para estabelecer as ligações:

- A coluna paciente_id (foreign key) referência a tabela **Paciente**, indicando qual paciente gerou ou recebeu essa senha (relações 1 paciente → N senhas).

- A coluna funcionario_id (foreign key) referência a tabela **Funcionário**, indicando qual profissional efetuou a chamada dessa senha (relações 1 funcionário → N senhas).

Portanto, cada registro em **Senha** está associado a exatamente um paciente e a exatamente um funcionário (1:N em ambas as direções).

➢ Tabela “Indicador”

A tabela **Indicador** reúne métricas gerenciais que o administrador pode consultar — por exemplo, tempo médio de espera, total de atendimentos, ou percentual de faltas. A chave primária é indicador_id (UUID). Os demais campos são tipo (String), apontando para que tipo de métrica se trata (por exemplo, “TempoMédioEspera” ou “TotalAtendimentos”), e valor (Float), que armazena o valor numérico da métrica naquele instante. Cada indicador carrega uma foreign key administrador_id apontando para quem criou ou consultou esse indicador. Assim, o relacionamento é:

- Um administrador (1) → pode ter zero ou muitos indicadores cadastrados ou consultados (0..*).

Cada indicador (muitos) pertence a um único administrador (1).

➢ Tabela “Notificação”

A entidade **Notificação** representa as mensagens enviadas aos pacientes, seja para avisar que sua senha foi chamada, que o tempo de resposta expirou ou que há uma nova atualização importante. A chave primária é notificacao_id (UUID). Os demais atributos são: tipo (String), que descreve o tipo de notificação (por exemplo, “ChamadaSenha” ou “Rechamada”), horario_envio (DateTime), que armazena o momento em que a notificação foi disparada, e status_entrega (String), com valores como “Pendente”, “Enviada” ou “Entregue”. No modelo, cada notificação possui duas foreign keys:

- senha_id, referenciando a tabela **Senha**, pois toda notificação é disparada automaticamente a partir de uma senha específica (1 senha → N notificações).

- paciente_id, referenciando a tabela **Paciente**, pois a notificação tem um destinatário, que é o paciente (1 paciente → N notificações).

Portanto, cada registro em **Notificação** está atrelado simultaneamente a uma senha e a um paciente, configurando duas relações do tipo “um para muitos”.

6. Descrição do software desenvolvido

• Visão geral do aplicativo:

- O SGFH é um sistema multiplataforma composto por um aplicativo Android para pacientes e funcionários, e um backend robusto construído com Spring Boot. O objetivo principal é modernizar a gestão da fila de atendimento, agendamentos e comunicação, oferecendo uma experiência mais ágil e transparente para todos os envolvidos.
- O sistema atende a quatro perfis de usuários distintos:
  1. **Paciente:** Acompanha sua posição na fila, agenda retornos e recebe notificações em tempo real.
  2. **Recepcionista:** Realiza o check-in dos pacientes, gerando senhas e inserindo-os na fila de atendimento.
  3. **Profissional de Saúde:** Visualiza a fila de espera, chama o próximo paciente e gerencia o fluxo de seu consultório.
  4. **Administrador:** Monitora os indicadores de desempenho da clínica através de um painel com gráficos e estatísticas.

• Arquitetura e Fluxo do Sistema

- O ecossistema do SGFH é baseado em uma comunicação clara entre o frontend (Dispositivo Android) e o backend (API REST), com o auxílio do Firebase Cloud Messaging (FCM) para notificações push, conforme ilustrado no diagrama de escopo.
- O fluxo geral opera da seguinte maneira:
  1. **Check-in (Recepcionista):** A recepcionista utiliza o app Android para fazer o check-in de um paciente. Isso envia uma requisição POST /checkin (implementado como POST /entradasAtendimento) para a API REST, que cria um novo registro na tabela attendance_entries do banco de dados.
  2. **Monitoramento (Paciente):** O paciente, através de seu aplicativo, visualiza sua posição na fila. O app faz uma requisição GET /fila/posicao (implementado como GET /entradasAtendimento) para obter a lista de espera e calcular sua posição.
  3. **Chamada (Profissional de Saúde):** O profissional de saúde, em seu painel, seleciona a opção para chamar o próximo paciente. O app envia um "payload" de notificação para a API REST através de uma requisição POST /notificacoes.
  4. **Notificação (FCM):** A API REST processa a requisição e, através do seu Serviço de Notificações, solicita ao Firebase Cloud Messaging (FCM) o envio de uma notificação push para o dispositivo do paciente.

5. **Confirmação (Paciente):** O paciente recebe a notificação e confirma sua presença. Essa ação dispara uma requisição POST /confirmacao (endpoint hipotético POST /entradasAtendimento/{id}/confirmar) para a API, atualizando o status do seu atendimento no banco de dados para CONFIRMADO.
6. **Outras Interações:** Pacientes também podem agendar retornos (POST /retornos), e administradores podem consultar indicadores (GET /indicadores).
7.

- Navegação do Aplicativo Android

  - O aplicativo foi estruturado para oferecer uma experiência de usuário intuitiva e segmentada por perfil.

    1. **Início e Login:** A jornada começa na SplashActivity , uma tela de boas-vindas que direciona o usuário para a MainActivity. Nela, o usuário escolhe entre o perfil de "Paciente" ou "Funcionário". A seleção leva às telas de login específicas: LoginPacienteActivity (com máscara de CPF) ou LoginFuncionarioActivity
    2. **Painel do Paciente (PacienteDashboardActivity):** Após o login, o paciente é direcionado a um painel com uma barra de navegação inferior (BottomNavigationView), que permite alternar entre os seguintes fragmentos:
       a. **Início/Fila (FilaFragment):** Tela principal onde o paciente vê sua posição na fila e o tempo de espera estimado.
       b. **Agendamento (AgendamentoFragment):** Permite ao paciente agendar e visualizar seus retornos, selecionando datas em um calendário e horários disponíveis.
       c. **Confirmação (ConfirmacaoFragment):** Este fragmento é tipicamente aberto após o recebimento de uma notificação de chamada, apresentando um cronômetro para o paciente confirmar sua presença.
       d. **Perfil (PerfilFragment):** Acessível pelo menu superior, permite ao paciente visualizar e editar seus dados cadastrais.
    3. **Painéis de Funcionários:** Dependendo do seu cargo (role), o funcionário logado é direcionado para a Activity correspondente:
       a. RecepcionistaActivity: Painel para check-in de pacientes, busca por CPF e geração de senha para uma especialidade.

b. ProfissionalSaudeActivity: Exibe a fila de pacientes aguardando atendimento. Possui um botão flutuante (FloatingActionButton) para chamar o próximo paciente, o que inicia a ChamarProximoActivity.

c. AdministradorActivity: Apresenta um dashboard com indicadores de performance, como tempo médio de espera e gráficos de atendimentos, utilizando a biblioteca MPAndroidChart.

- API REST e Backend (Spring Boot):

  - O backend é o cérebro do sistema, centralizando todas as regras de negócio e a persistência de dados.

    1. **Controladores:** A API expõe endpoints RESTful para cada entidade do sistema, como PacienteController , AttendanceEntryController , FollowUpController e NotificationController. A autenticação é gerenciada pelo AuthController.

    2. **Serviços:** A lógica de negócio é encapsulada em classes de serviço (PacienteService , NotificationService, etc.), que orquestram as operações entre os controladores e os repositórios de dados.

    3. **Segurança:** O sistema utiliza Spring Security para proteger os endpoints. Os endpoints de login (/api/auth/**) são públicos, enquanto os demais podem ser configurados para exigir autenticação. As senhas são armazenadas de forma segura usando BCryptPasswordEncoder.

    4. **Integração com Firebase:** A classe FirebaseConfig estabelece a conexão com o projeto Firebase, permitindo que o NotificationService envie notificações push de forma eficiente.

- Banco de Dados (PostgreSQL)

  - O sistema é sustentado por um banco de dados relacional PostgreSQL, escolhido por sua robustez e recursos avançados. O diagrama de escopo menciona SQLite, mas o script de banco de dados fornecido é claramente para PostgreSQL, uma escolha mais adequada para um ambiente de servidor.

  - **Estrutura de Tabelas:** O banco é modelado com tabelas principais que refletem as entidades do sistema:

    1. patients: Armazena os dados cadastrais completos dos pacientes.

    2. employees: Contém as informações dos funcionários, incluindo sua matrícula e cargo (role).

    3. specialties: Tabela de especialidades médicas.

4. attendance_entries: Tabela central que gerencia a fila, ligando um paciente (patient_id) a uma especialidade (specialty_id) com um status e timestamps relevantes.
5. follow_ups: Registra os agendamentos de retorno dos pacientes.
6. device_tokens: Armazena os tokens de dispositivo (FCM) para cada paciente, permitindo o envio de notificações.
7. **Integridade de Dados:** O esquema utiliza tipos de dados específicos como ENUM (attendance_status, followup_status, employee_role) e TIMESTAMPTZ (timestamp com fuso horário) para garantir a consistência e precisão dos dados. Além disso, índices foram criados em campos-chave como cpf e matricula para otimizar as consultas.

- Melhorias Futuras

  - **Integração com WhatsApp:** Implementar um serviço de notificação via WhatsApp como uma alternativa ou complemento às notificações push. Isso garante que pacientes que não tenham o aplicativo instalado ou estejam com as notificações desativadas ainda possam ser contatados para chamadas e lembretes de agendamento.
  - **Sistema de Pulseiras de Chamada:** Para pacientes que não possuem smartphone ou têm dificuldades com a tecnologia, será desenvolvido um sistema de pulseiras com identificação (via QR Code ou RFID). A recepção vincularia a pulseira ao cadastro do paciente no momento do check-in. Quando chamado, um painel central na recepção ou monitores espalhados pela clínica exibiriam o número ou código da pulseira, garantindo que ninguém perca sua vez.

7. Conclusão

- Em resumo, o projeto SGFH demonstra ser uma solução de software completa e bem arquitetada, que aborda de forma eficaz os desafios do gerenciamento de fluxo em clínicas e hospitais. Com perfis de usuário bem definidos, uma navegação coesa no aplicativo Android, e um backend seguro e funcional, o sistema está pronto para otimizar a experiência de pacientes e funcionários, reduzindo tempos de espera e melhorando a comunicação

- As melhorias futuras propostas mostram um caminho claro para tornar o sistema ainda mais inclusivo e robusto, solidificando seu valor como uma ferramenta essencial para a modernização da gestão em saúde.

- Obrigado pela sua atenção!