

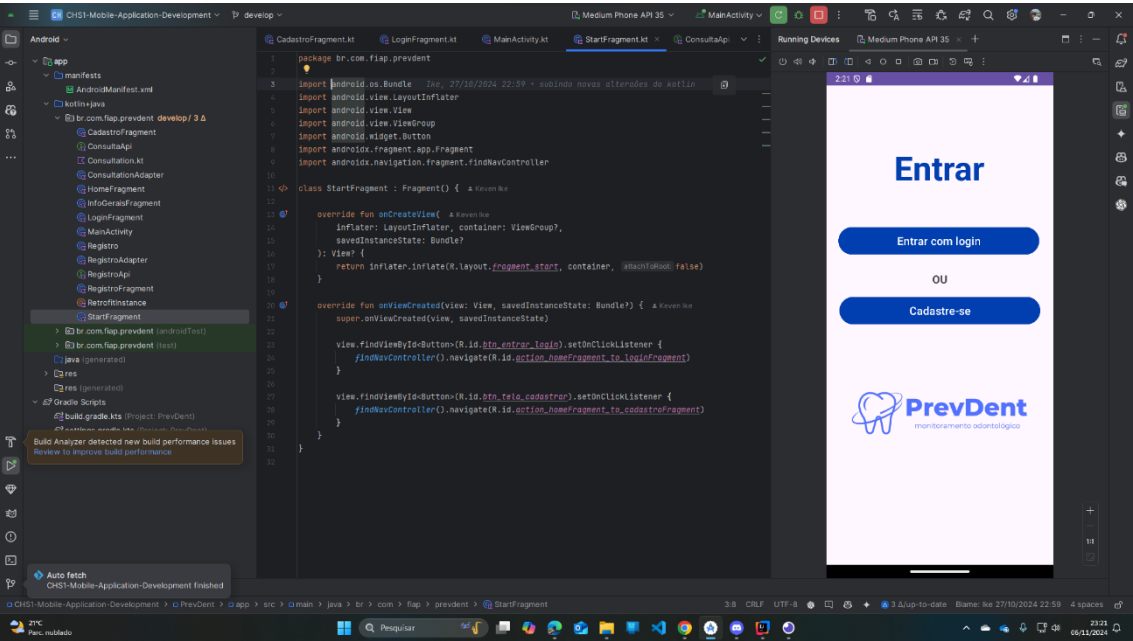
Documentação do Projeto PrevDent

Alunos	RM
Keven Ike	553215
Vitor Santos	553621
Jose Neto	553844

1. Introdução ao Projeto

Nome do Projeto: PrevDent

Descrição Geral: PrevDent é um aplicativo desenvolvido para o monitoramento odontológico, proporcionando uma plataforma intuitiva para que os usuários registrem sintomas, acompanhem consultas odontológicas agendadas e visualizem informações essenciais sobre a saúde bucal. A finalidade do aplicativo é otimizar o controle odontológico, oferecendo acesso rápido e eficiente a dados pertinentes tanto para pacientes quanto para profissionais da área da saúde, promovendo uma abordagem proativa no cuidado odontológico.

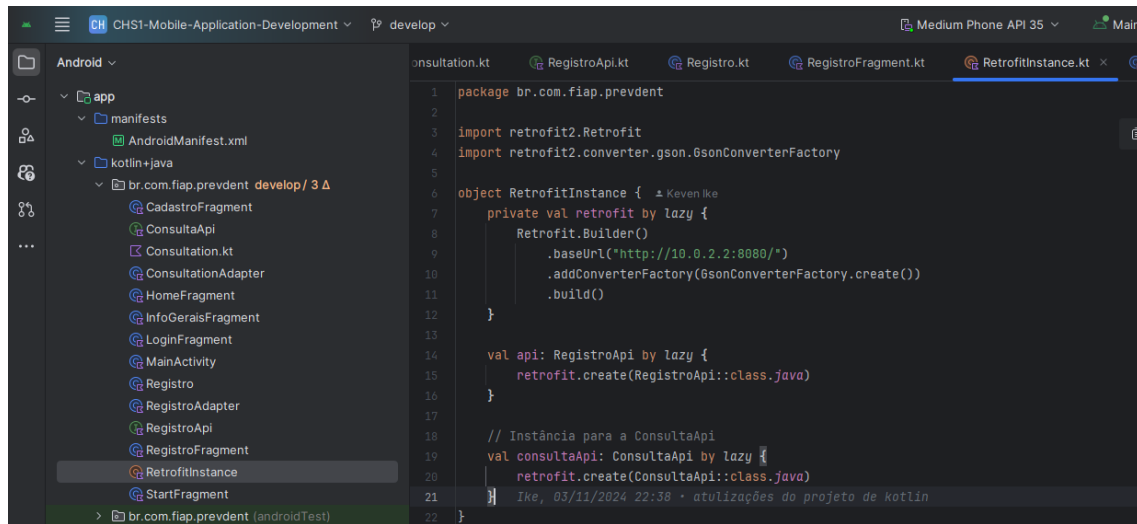


2. Arquitetura do Projeto

Componentes Principais: A arquitetura do projeto foi organizada em fragmentos, possibilitando a modularidade e a manutenção simplificada das diferentes telas do aplicativo, tais como HomeFragment, LoginFragment e RegistroFragment. A navegação entre essas telas é orquestrada pelo Navigation Component (versão

2.8.3), que assegura uma experiência fluida ao usuário, reduzindo a complexidade do gerenciamento de navegação dentro do ciclo de vida dos fragmentos.

Comunicação com Backend: A comunicação com o backend é implementada utilizando Retrofit, uma biblioteca que facilita a integração com APIs REST, fornecendo uma interface intuitiva e suporte à conversão de dados. A classe `RetrofitInstance.kt` realiza a configuração do Retrofit, definindo a `baseUrl` e instanciando as interfaces das APIs que manipulam os dados de consulta e registro, permitindo o fluxo seguro e eficiente de informações entre o cliente e o servidor.

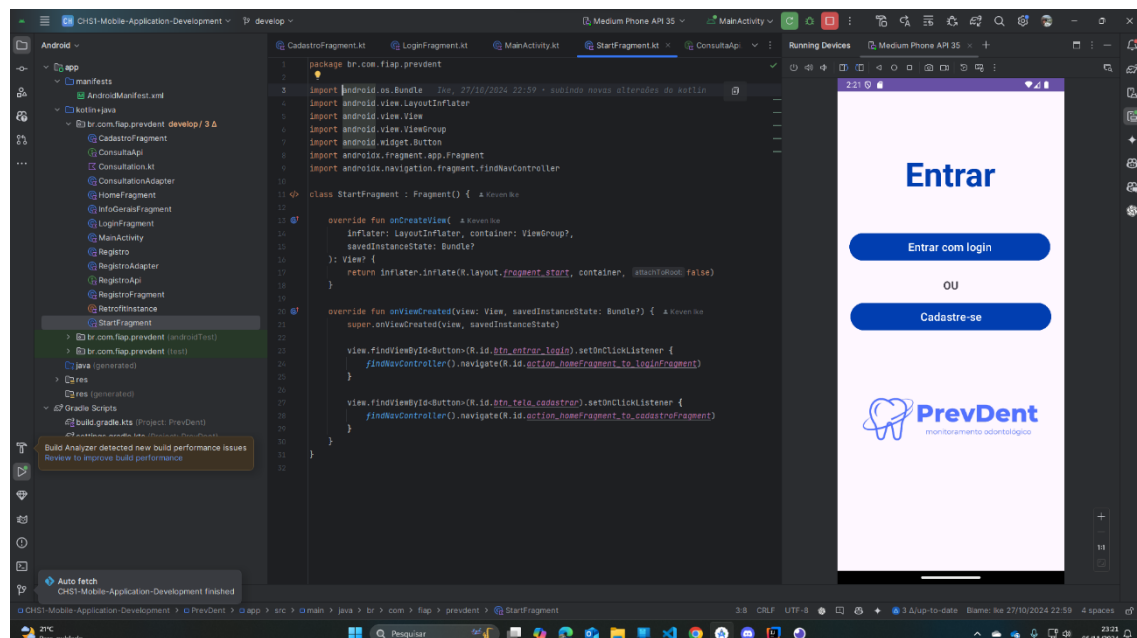


```
1 package br.com.fiap.prevdent
2
3 import retrofit2.Retrofit
4 import retrofit2.converter.gson.GsonConverterFactory
5
6 object RetrofitInstance {
7     private val retrofit by lazy {
8         Retrofit.Builder()
9             .baseUrl("http://10.0.2.2:8080/")
10            .addConverterFactory(GsonConverterFactory.create())
11            .build()
12        }
13
14        val api: RegistroApi by lazy {
15            retrofit.create(RegistroApi::class.java)
16        }
17
18        // Instância para a ConsultaApi
19        val consultaApi: ConsultaApi by lazy {
20            retrofit.create(ConsultaApi::class.java)
21        }
22    }
```

3. Funcionalidades

Tela Inicial (StartFragment)

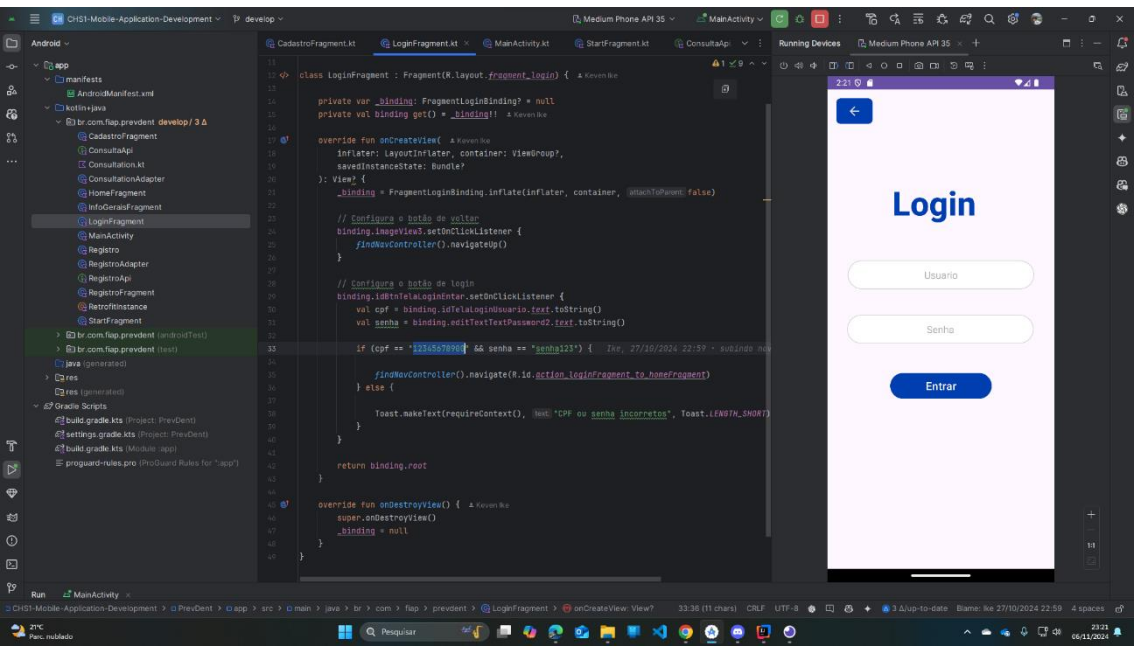
A tela inicial serve como ponto de entrada do usuário no aplicativo, oferecendo duas opções principais: **Entrar com login** ou **Cadastrar-se**. A interface direciona o usuário conforme suas credenciais, garantindo uma experiência personalizada de acordo com seu perfil, seja ele um novo usuário ou um usuário recorrente.



```
1 package br.com.fiap.prevdent
2
3 import androidx.fragment.app.Fragment
4 import androidx.navigation.fragment.NavHostController
5
6 class StartFragment : Fragment() {
7     override fun onCreateView(
8         inflater: LayoutInflater, container: ViewGroup?,
9         savedInstanceState: Bundle?
10     ): View? {
11         return inflater.inflate(R.layout.fragment_start, container, false)
12     }
13
14     override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
15         super.onViewCreated(view, savedInstanceState)
16
17         view.findViewById<Button>(R.id.btn_entrar_login).setOnClickListener {
18             findNavController().navigate(R.id.action_homeFragment_to_loginFragment)
19         }
20
21         view.findViewById<Button>(R.id.btn_tela_cadastro).setOnClickListener {
22             findNavController().navigate(R.id.action_homeFragment_to_cadastroFragment)
23         }
24     }
25 }
```

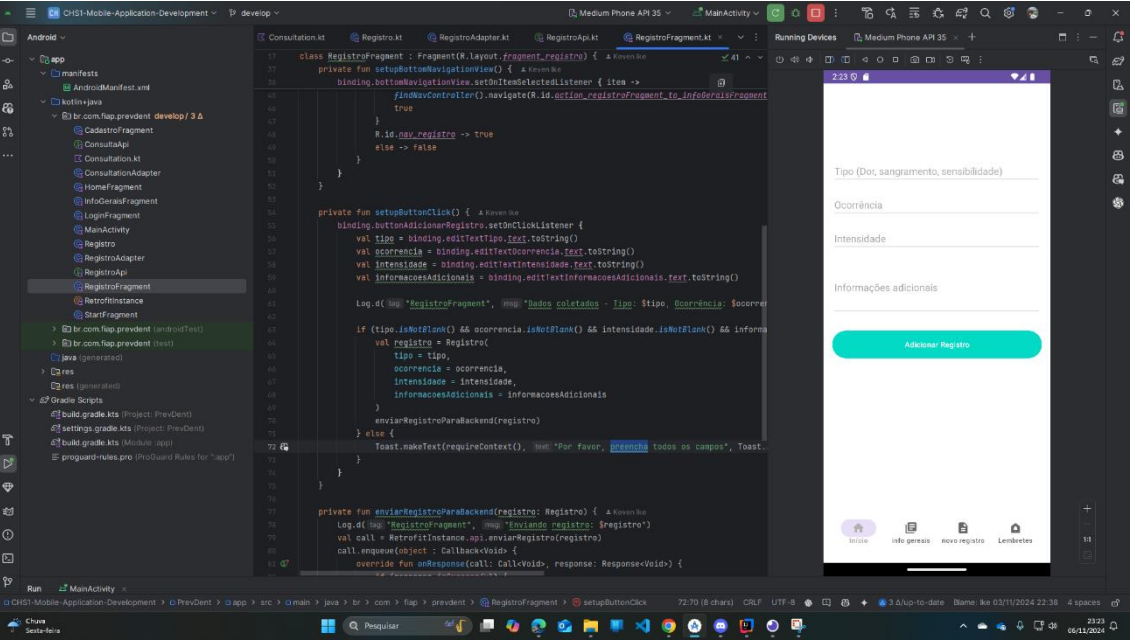
Tela de Login (LoginFragment)

Na tela de login, o usuário deve inserir o CPF e a senha para autenticação. Este processo é facilitado por uma validação local que verifica a conformidade das credenciais antes de permitir o acesso às funcionalidades principais do aplicativo. A validação atual utiliza dados fictícios como exemplo, que devem ser substituídos por um sistema de autenticação seguro em produção.



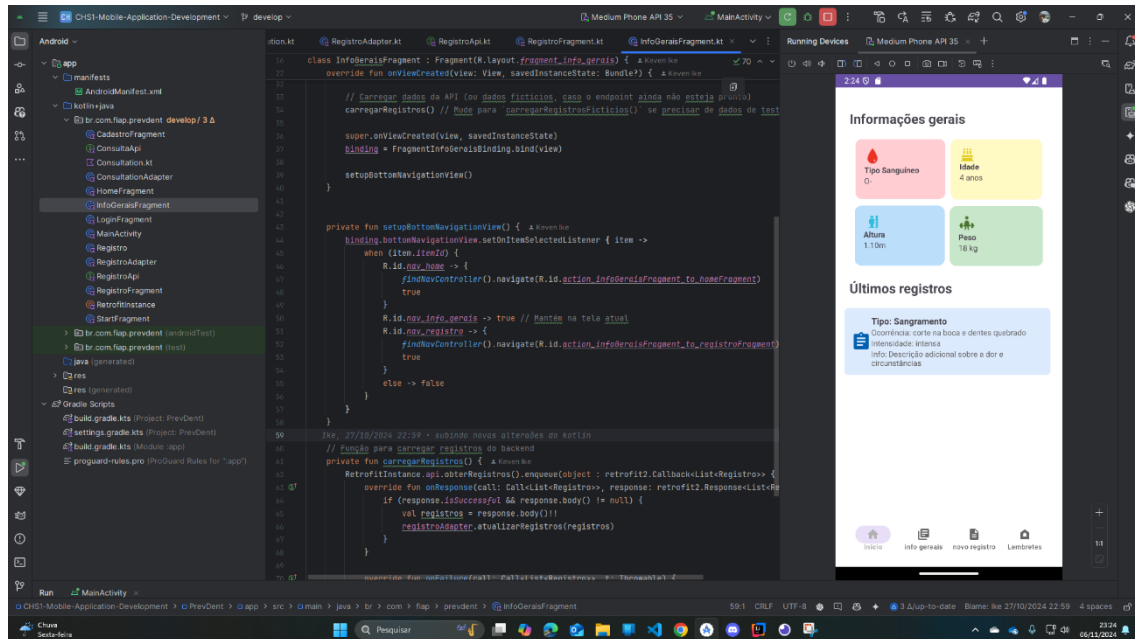
Tela de Registro de Sintomas (RegistroFragment)

A tela de registro de sintomas permite ao usuário documentar sintomas odontológicos que estejam experienciando. O formulário captura informações detalhadas, como tipo do sintoma (por exemplo: dor ou sangramento), ocorrência, intensidade, e informações adicionais pertinentes. Esses dados são enviados ao backend por meio do Retrofit, onde são processados e posteriormente exibidos na tela de "Informações Gerais" para acompanhamento contínuo.



Tela de Informações Gerais (InfoGeraisFragment)

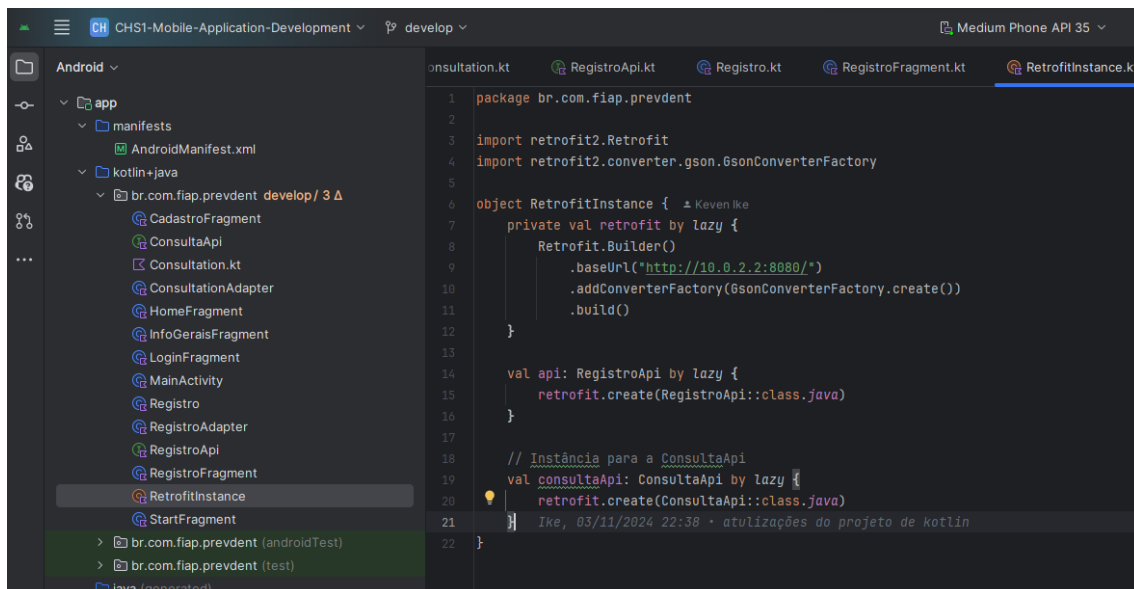
Nesta tela, o usuário tem acesso a um resumo consolidado de suas informações de saúde, incluindo dados como tipo sanguíneo, idade, altura e peso. A seção "Últimos Registros" apresenta um histórico dos sintomas registrados, incluindo detalhes críticos sobre a ocorrência e intensidade, promovendo um monitoramento contínuo e uma melhor comunicação com os profissionais de saúde.



4. Estrutura de Código

Classes Principais

- **Consultation.kt:** Esta classe modela as consultas do paciente, contendo atributos que descrevem o dentista responsável, o paciente, o tipo de tratamento e o diagnóstico. Cada instância dessa classe representa um evento clínico associado ao histórico do paciente.
- **Registro.kt:** Define o modelo para armazenar registros de sintomas, estruturando as informações inseridas pelo usuário, tais como tipo de sintoma, ocorrência, intensidade, e observações adicionais, garantindo consistência nos dados transmitidos ao backend.
- **RetrofitInstance.kt:** Responsável por configurar o Retrofit, incluindo a definição da baseUrl e a criação das instâncias das interfaces de API que serão usadas para a comunicação com o servidor. Esta classe segue o padrão Singleton para assegurar que apenas uma instância do Retrofit seja usada em todo o aplicativo, otimizando recursos.



5. Detalhes Técnicos

Tecnologias Utilizadas: O projeto foi desenvolvido utilizando Kotlin como linguagem principal, Retrofit para comunicação com o backend, Navigation Component para gerenciamento de navegação entre as telas, e Android Studio como ambiente de desenvolvimento integrado (IDE).

Integração Backend: A integração com o backend é mediada pelo Retrofit, que realiza as requisições REST para os endpoints definidos pelas interfaces RegistroApi e ConsultaApi. Essas interfaces descrevem os métodos HTTP utilizados para registrar sintomas e obter informações de consultas, permitindo um fluxo de dados coerente e seguro.

Anotações: Para facilitar a serialização e desserialização dos dados trocados com o backend, as classes de dados utilizam a anotação @SerializedName, garantindo que os nomes dos campos correspondam aos especificados na API, mesmo quando o nome das variáveis no código diverge do padrão exigido pelo backend.

```
5 data class Consultation(  Keven Ike
6     @SerializedName("paciente") val paciente: PacienteInfo,
7     @SerializedName("dentista") val dentista: DentistaInfo,
8     @SerializedName("data") val data: String,
9     @SerializedName("tipoTratamento") val tipoTratamento: String,
10    @SerializedName("diagnostico") val diagnostico: DiagnosticoInfo
11 )
12
13
14 data class PacienteInfo(  Keven Ike
15     @SerializedName("nome") val nome: String
16 )
17
18 data class DentistaInfo(  Keven Ike
19     @SerializedName("nome") val nomeDentista: String
20 )
21
22 data class DiagnosticoInfo(  Keven Ike
23     @SerializedName("descricao") val descricao: String
24 )
```