INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS

Departamento De Engenharias e Tecnologias

APLICAÇÕES MÓVEIS

MEMORANDO DO LABORATÓRIO 4

Docente:

_____

João Costa

CONTACTOS

Wilker Matias - 20210053@isptec.co.ao

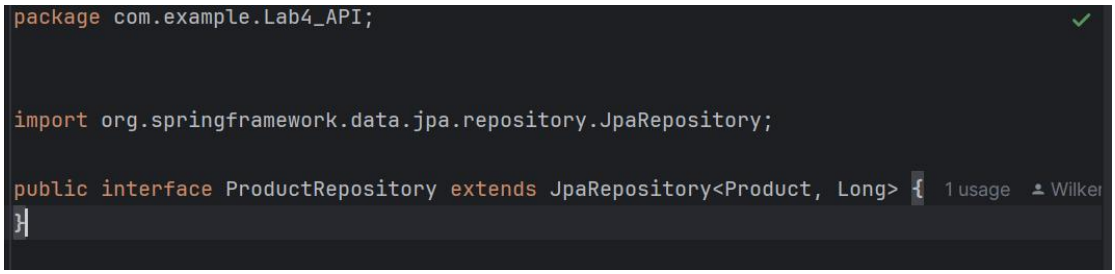
IMPLEMENTAÇÃO DA API NO INTELLIJ

Para a implementação da API comecei por criar o projecto usei o Spring initializr para a criação do projecto com as dependências sendo o “Spring web”, “Spring data jpa” e o “MySQL driver”. Após isso criou-se a classe “Product” com os seus atributos sendo “id” e “name”, os seus getters e setters e defini-a como sendo entidade que será usada na base de dados usando @Entity (Não foi necessário o uso da @Table pois o nome da classe foi o mesmo que o nome da tabela na base de dados).

```java
package com.example.Lab4_API;



import jakarta.persistence.*;

@Entity  5 usages  WilkerMatias
public class Product {
    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;  2 usages

    public Long getId() {  no usages  WilkerMatias
        return id;
    }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

}
```

Em seguida foi criado o repositório da classe “Product”, a interface “ProductRepository” que estende de JpaRepository, que é uma interface fornecida pelo Spring Data JPA. Isso permite que ProductRepository herde uma série de

métodos prontos para o uso, como save(), findById(), findAll(), delete(), e outros, que simplificam as operações CRUD (Create, Read, Update, Delete) na entidade Product.



Por fim, criou-se a classe “ProductController” responsável por gerenciar as requisições HTTP que interagem com a entidade “Product”. Nela usei a anotação @RestController que indica que esta classe é um controlador Spring e a anotação @RequestMapping("/products") que define que a base do caminho das requisições que o controlador gerencia, especificando que todos os endpoints serão acessíveis a partir da URL /products.

Os endpoints definidos foram:

- GET /products usando o método getAllProducts() junto da anotação @GetMapping: que retorna uma lista de todos os produtos armazenados no banco de dados.
- POST /products usando o método createProduct() junto da anotação @PostMapping: que recebe um objeto “Product” no corpo da requisição e o salva no banco de dados.
- GET /products/{id}: usando o método getProductById() junto da anotação @GetMapping("/{id}"): que faz a busca de um produto específico pelo seu ID.

```java
package com.example.Lab4_API;                                    ⚠ 5

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController  no usages  WilkerMatias
@RequestMapping("/products")
public class ProductController {

    @Autowired  3 usages
    private ProductRepository productRepository;

    @GetMapping  no usages  WilkerMatias
    public List<Product> getAllProducts() { return productRepository.findAll(); }

    @PostMapping  no usages  WilkerMatias
    public Product createProduct(@RequestBody Product product) { return productReposit

    @GetMapping("/{id}")  no usages  WilkerMatias
    public Product getProductById(@PathVariable Long id) { return productRepository.fi
}
```

IMPLEMENTAÇÃO DA INTERFACE NO ANDROID STUDIO

MainActivity

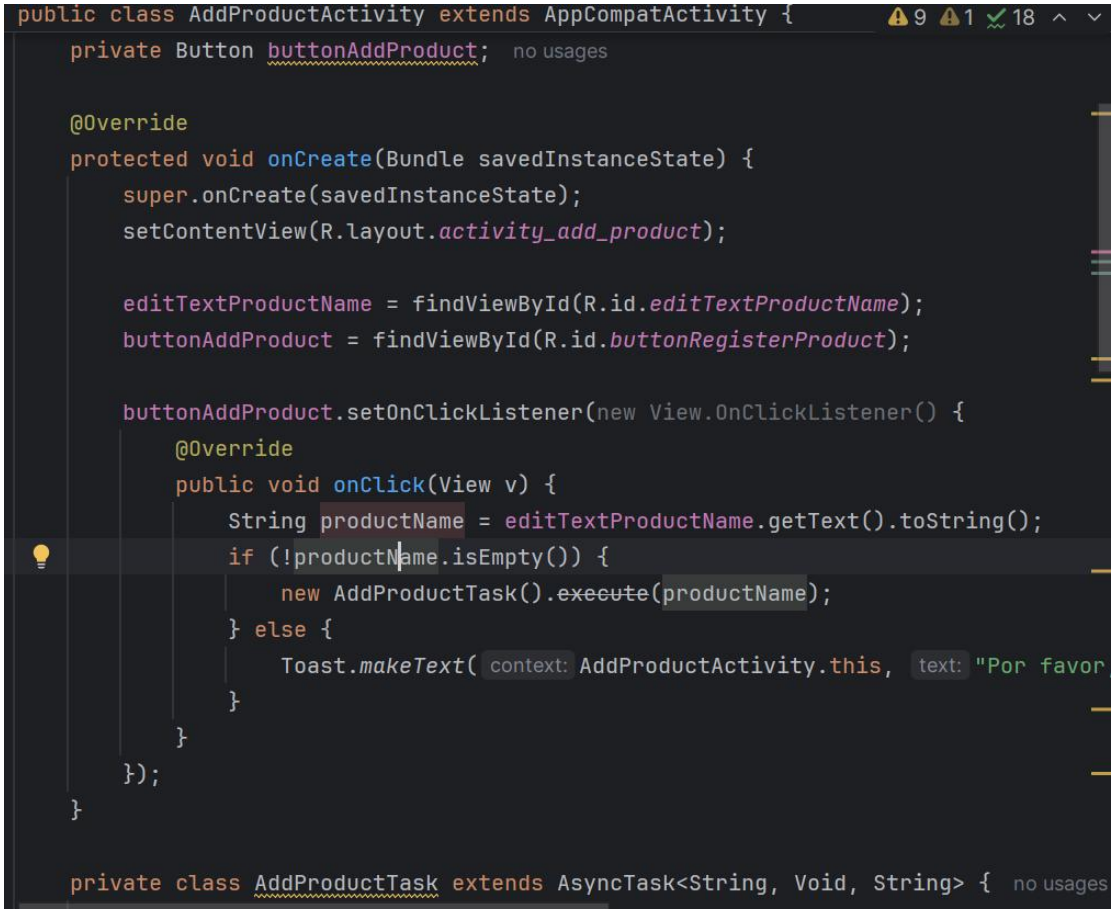
A MainActivity é a tela inicial da aplicação, onde o usuário pode escolher entre duas opções principais: registrar um novo produto ou buscar produtos existentes. É composta por dois botões: “buttonRegisterProduct” e “buttonSearchProduct”.

- buttonRegisterProduct(Cadastrar produto): redireciona o usuário para a AddProductActivity para que o usuário possa adicionar um novo produto.
- buttonSearchProduct(Pesquisar produto): redireciona o usuário para a SearchProductActivity para buscar produtos cadastrados.

AddProductActivity

A AddProductActivity é responsável por registrar novos produtos no sistema, enviando os dados para a API através de uma requisição HTTP POST. Ela composta por um editText que espera a entrada de dados do usuário (o nome do produto) e um botão (Adicionar produto), assim que clicado o botão verifica se o campo de texto não está vazio antes de iniciar a requisição. Caso ele verifique que o campo não está vazio ele inicia a requisição pelo “AddProductTask” que é uma AsyncTask para realizar operações de rede em segundo plano. Nela são acionados os seguintes métodos:

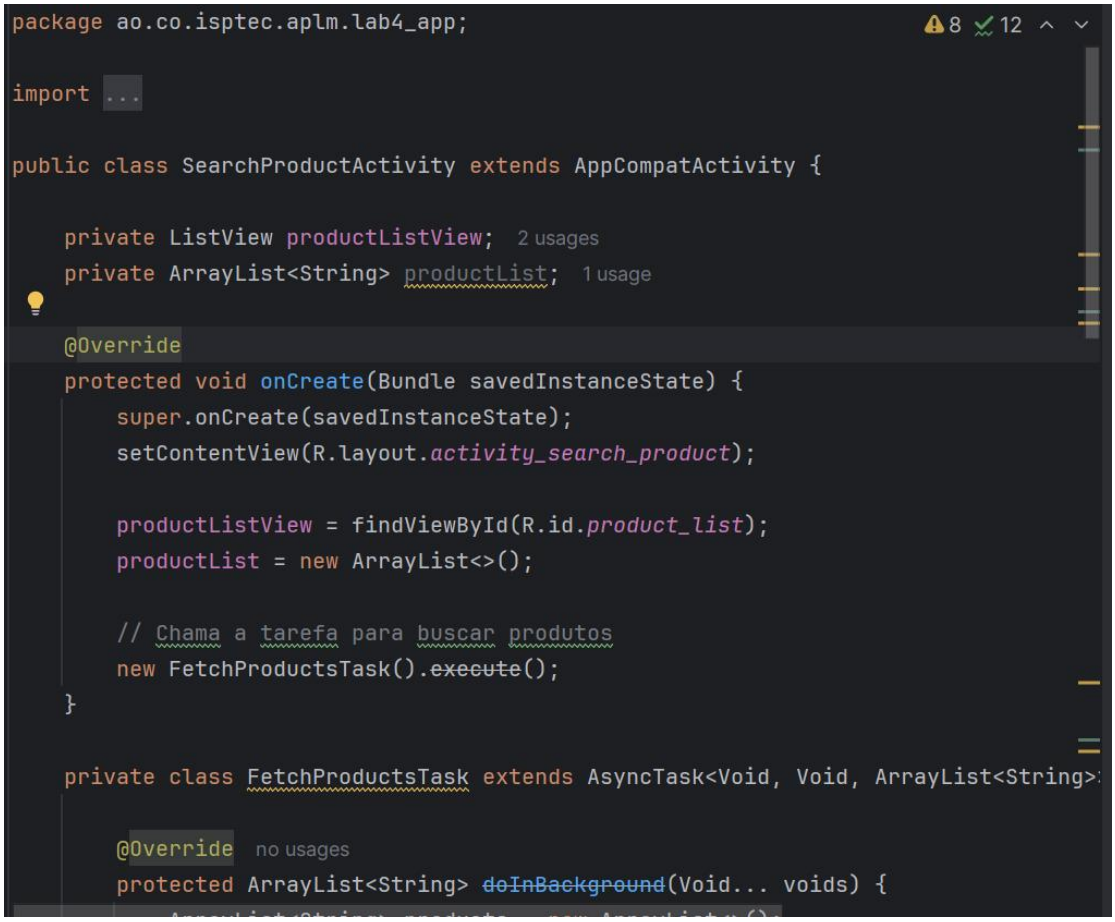
- doInBackground(): envia uma requisição POST para a API, incluindo o nome do produto no corpo da requisição como um objeto JSON.
- onPostExecute(): exibe uma mensagem de confirmação usando Toast e limpa o campo de texto.



SearchProductActivity

A SearchProductActivity é projetada para buscar e listar produtos já cadastrados, exibindo-os em uma “ListView”. Tal como no “AddProductActivity” foi definida uma classe privada “FetchProductsTask” que é uma AsyncTask.

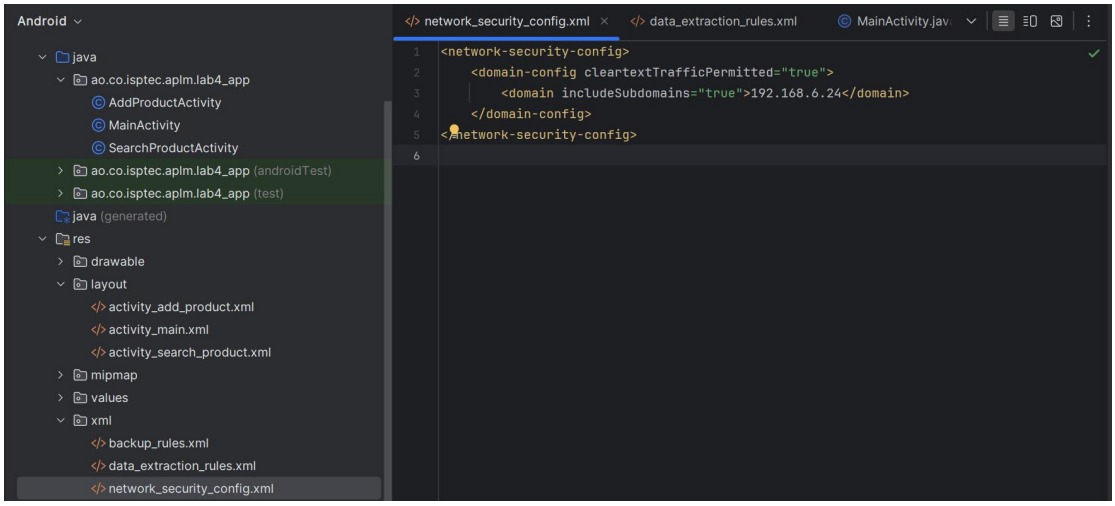
- doInBackground(): Realiza a requisição GET para a API e processa a resposta JSON, extraindo os nomes dos produtos.
- onPostExecute(): atualiza a ListView com os nomes dos produtos.



```
package ao.co.isptec.aplm.lab4_app;                                    8 12

import ...

public class SearchProductActivity extends AppCompatActivity {

    private ListView productListView;  2 usages
    private ArrayList<String> productList;  1 usage

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_search_product);

        productListView = findViewById(R.id.product_list);
        productList = new ArrayList<>();

        // Chama a tarefa para buscar produtos
        new FetchProductsTask().execute();
    }

    private class FetchProductsTask extends AsyncTask<Void, Void, ArrayList<String>>

        @Override  no usages
        protected ArrayList<String> doInBackground(Void... voids) {
```

Configurações adicionais

Finalmente, após configurar as actividades, dentro da pasta “res/xml” criou-se o fichiero “network_security_config” para definir regras de segurança de rede para a aplicação Android, permitindo o tráfego de dados em texto claro (sem criptografia) entre a aplicação e a API Spring Boot.



O endereço 192.168.6.24 corresponde ao endereço do meu computador na rede, de forma a que a aplicação Android possa se conectar corretamente à API Spring Boot que está rodando localmente. Isso permite que a aplicação envie requisições HTTP para o backend e receba as respostas.

E para terminar fez-se a actualização do ficheiro “AndroidManifest.xml” para que o aplicativo tenha acesso a internet usando o “<uses-permission android:name="android.permission.INTERNET" />”, para que possa usar tráfego de rede não criptografado (HTTP) usando o “ android:usesCleartextTraffic="true"” e define quais domínios podem ser acessados pelo aplicativo, especificando que o tráfego de rede é permitido para determinados endereços.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:networkSecurityConfig="@xml/network_security_config"
        android:usesCleartextTraffic="true"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Lab4_APP"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Lab4_APP"
        tools:targetApi="31">
        <activity
            android:name=".SearchProductActivity"
            android:exported="false" />
        <activity
            android:name=".AddProductActivity"
```