Relatório Técnico da Aplicação ClothingStore

Ângelo Neto [1230439], Luís Oliveira [1231002]

Instituto Superior de Engenharia do Porto {1230439,Anglnet,
1231002,1231002Luis}@github.com
https://github.com/1231002Luis/DSSMV_ProjectDroid_1230439_1231002

Abstract. This report describes the development of an Android-based application, using Java as the programming language , named "ClothingStore," a digital clothing store designed to offer users an interactive and efficient platform for browsing, filtering, and purchasing apparel. The app allows users to search for clothing items across various categories, apply filters, and view product details, including description and prices. The user can create an account, and add articles from the store to his cart in order to buy them.
To deliver real-time and accurate data, "ClothingStore" integrates a REST API that manages communication with the backend, facilitating seamless access to up-to-date information on inventory, pricing, and user-generated reviews. This API-driven architecture enables scalable and efficient data handling, enhancing the app's responsiveness and ensuring a consistent experience across devices.

Key words: Android , Java, digital clothing store, browsing , filtering, prices, reviews, REST API, backend

1 Introdução

Este relatório apresenta o desenvolvimento do aplicação "ClothingStore," uma plataforma de loja de roupas digital voltada para dispositivos Android. Desenvolvido em Java, a aplicação foi projetada para proporcionar uma experiência de compra otimizada, permitindo que os usuários pesquisem e explorem uma ampla variedade de peças de vestuário. A "ClothingStore" oferece funcionalidades de filtragem por categoria, nome e tipo de peça, além de exibir detalhes sobre a peça e o seu preço.

A aplicação usa uma arquitetura baseada em REST API para gerir a comunicação com o backend, garantindo que as informações apresentadas estejam sempre atualizadas e sejam recuperadas de maneira eficiente. Esta abordagem permite uma gestão de dados em tempo real e uma experiência de navegação fluida, com respostas rápidas e confiáveis aos comandos dos usuários. A escolha da linguagem Java para o desenvolvimento da "ClothingStore" visa garantir um desempenho estável e uma fácil manutenção do código, aproveitando a compatibilidade e robustez do ecossistema Android.

Este relatório aborda a análise e o design da aplicação. Além disso, são discutidos aspectos de design de interface e testes.

2 Análise

2.1 Modelo de domínio

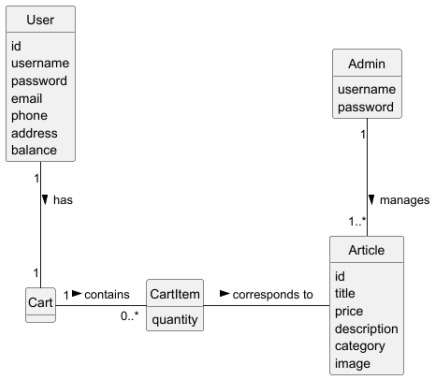


Fig. 1. Modelo de domínio

O modelo de domínio da nossa aplicação consiste apenas em 3 classes conceptuais: User, Article e CartItem. Na nossa aplicação, o user através da sua conta pode adicionar artigos ao seu carrinho para os comprar. Cada utilizador pode ter vários items no seu carrinho com uma determinada quantidade, e esse item corresponde a um determinado artigo presente na loja.

2.2 Requisitos não funcionais

- Usabilidade
  - Interface gráfica.
- Fiabilidade
  - Não especificado.
- Desempenho
  - Deve ocupar pouca memória.
  - Consumo de bateria baixo.
- Sustentabilidade
  - Plataforma android.
  - API android 21
- Restrições de Design
  - Não especificado.
- Restrições de Implementação
  - Linguagem Java.
- Restrições de Interface
  - Interface gráfica android.

2.3 Requisitos funcionais

– Funcionalidades

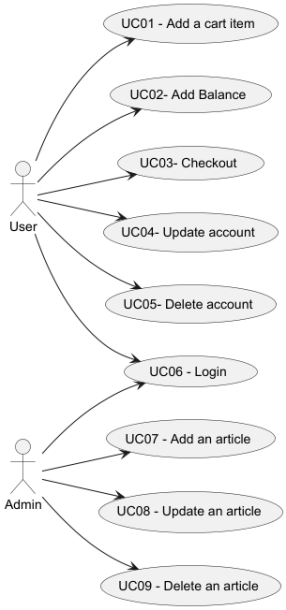


Fig. 2. Diagrama de casos de uso

Foram estabelecidos os seguintes casos de uso:

– User
  - UC01- Adicionar artigo ao carrinho
  - UC02 - Adicionar saldo.
  - UC03 - Checkout.
  - UC04 - Atualizar conta.
  - UC05 - Apagar conta.
  - UC06 - Login.

– Admin
  - UC06- Login.
  - UC07 - Adicionar artigo.
  - UC08 - Atualizar artigo.
  - UC09 - Apagar artigo.

2.4 Especificação UC01

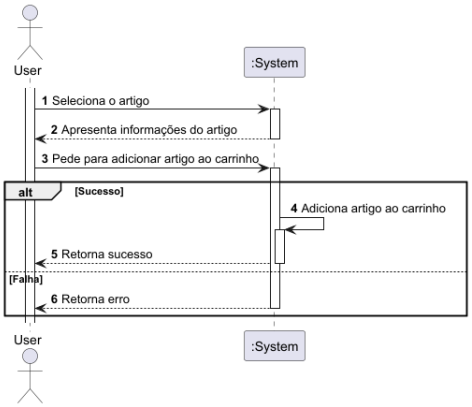


Fig. 3. SSD-UC01 : Adicionar artigo ao carrinho

Descrição	Adicionar artigo ao carrinho.
Pré-condição	Ter sessão de user iniciada e estar na página do artigo a adicionar.
Pós-condição	O artigo é adicionado ao carrinho.
Caminho principal	1. O user seleciona o artigo. 2. O sistema apresenta as informações do artigo. 3. O user pede para adicionar o artigo ao carrinho. 4. O sistema adiciona o artigo ao carrinho. 5. O sistema retorna sucesso.
Caminho alternativo	1. Falha ao adicionar artigo ao carrinho. 2. Retorna erro.

2.5 Especificação UC02

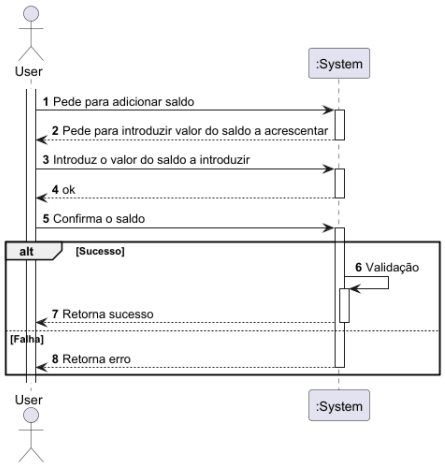


Fig. 4. SSD-UC02 : Adicionar saldo

Descrição	Adicionar saldo.
Pré-condição	Ter sessão de user iniciada e estar na página de informações do user.
Pós-condição	O saldo é adicionado.
Caminho principal	1. O user pede para adicionar saldo. 2. O sistema pede para introduzir valor a acrescentar. 3. O user introduz valor do saldo a introduzir e confirma. 4. O saldo é adicionado. 5. O sistema retorna sucesso.

Caminho alternativo	1. Saldo a introduzir superior a 1000. 2. Retorna erro.

2.6 Especificação UC03

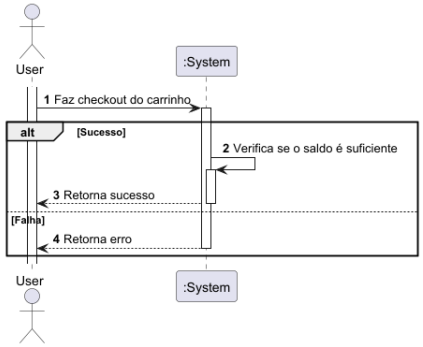


Fig. 5. SSD-UC03 : Checkout

Descrição	Checkout do carrinho.
Pré-condição	Ter sessão de user iniciada e estar na página do carrinho.
Pós-condição	Checkout é feito com sucesso.
Caminho principal	1. O user pede para fazer checkout. 2. O sistema verifica se o saldo é suficiente. 3. O sistema retorna sucesso.
Caminho alternativo	1. O saldo não é suficiente. 2. Retorna erro.

2.7 Especificação UC04

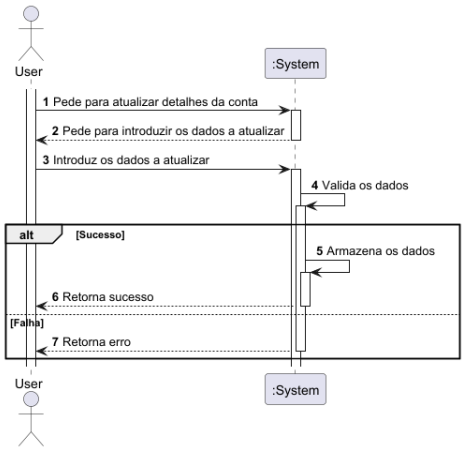


Fig. 6. SSD-UC04 : Atualizar conta

Descrição	Atualizar dados da conta.
Pré-condição	Ter sessão de user iniciada e estar na página de informações do user
Pós-condição	Alteração dos dados é efetuada.
Caminho principal	1. O user pede para atualizar detalhes da conta. 2. O sistema pede para introduzir os dados a atualizar. 3. O user introduz todos os dados. 4. O sistema valida os dados. 5. O sistema armazena os dados. 6. Retorna sucesso.

Caminho alternativo	1. Os dados são inválidos ou já existem. 2. Retorna erro.

2.8 Especificação UC05

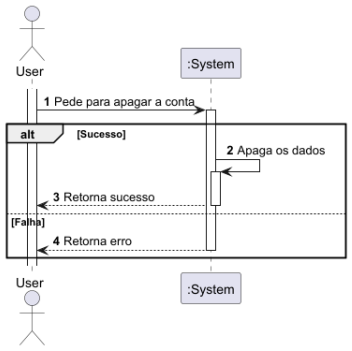


Fig. 7. SSD-UC05 : Apagar conta

Descrição	Apagar a conta de user
Pré-condição	Ter sessão de user iniciada e estar na página de informações do user
Pós-condição	Conta de usar apagada
Caminho principal	1. O user pede para apagar a conta 2. O sistema remove a conta. 3. Retorna sucesso.
Caminho alternativo	1. Retorna erro.

2.9 Especificação UC06

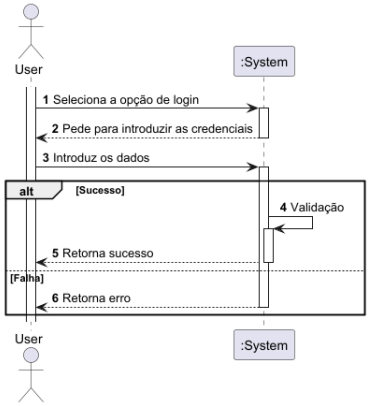


Fig. 8. SSD-UC06 : Login

Descrição	Fazer login na conta
Pré-condição	Estar na página de login
Pós-condição	Login efetuado
Caminho principal	1. O utilizador seleciona a opção de login 2. O sistema pede as credenciais 3. O utilizador introduz as credenciais 4. O sistema valida a conta e averigua se existe 5. Retorna sucesso.
Caminho alternativo	1. As credenciais estão erradas ou conta não existe. 2. Retorna erro.

2.10 Especificação UC07

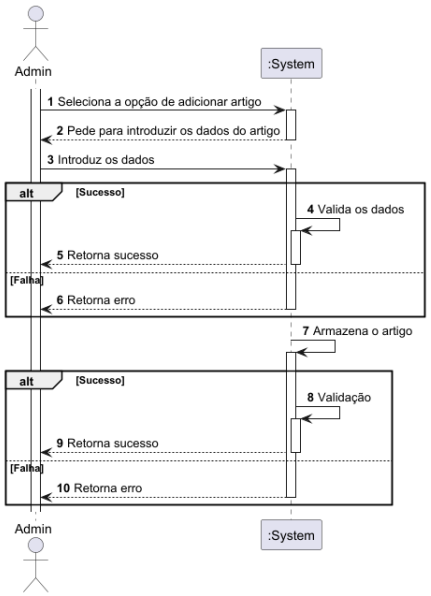


Fig. 9. SSD-UC07 : Adicionar artigo

Descrição	Adicionar artigo à loja
Pré-condição	Ter sessão de administrador iniciada
Pós-condição	Artigo adicionado à loja
Caminho principal	1. O administrador pede para adicionar artigo 2. O sistema pede os dados do artigo 3. O adminsitrador introduz os dados 4. O sistema valida os dados 5. Retorna sucesso 6. O sistema armazena o artigo 7. O sistema valida o artigo 8. Retorna sucesso

Caminho alternativo	1. Dados incompletos ou incompatíveis – O sistema pede ao administrador para reintroduzir os dados 2. Artigo já existente – Retorna erro

2.11 Especificação UC08

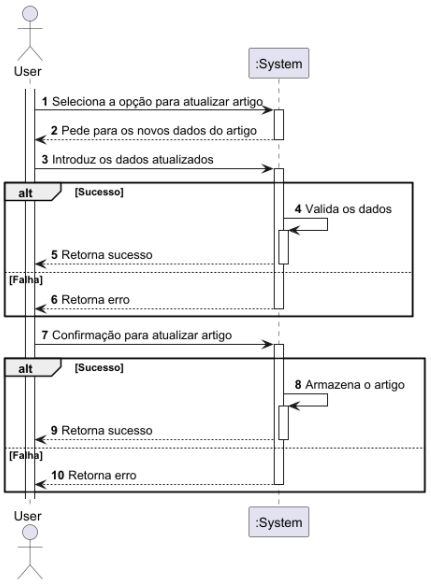


Fig. 10. SSD-UC08 : Atualizar artigo

Descrição	Atualizar dados do artigo
Pré-condição	Ter sessão de administrador iniciada e ter produto a atualizar selecionado
Pós-condição	Dados do artigo atualizados
Caminho principal	1. O administrador seleciona a opção para atualizar o artigo 2. O sistema pede os novos dados do artigo 3. O adminsitrador introduz os dados 4. O sistema valida os dados 5. Retorna sucesso 6. Administrador confirma atualização 7. O sistema armazena o artigo 8. Retorna sucesso
Caminho alternativo	1. Dados incompletos ou incompatíveis – O sistema pede ao administrador para reintroduzir os dados 2. Erro ao guardar artigo – Retorna erro

2.12 Especificação UC09

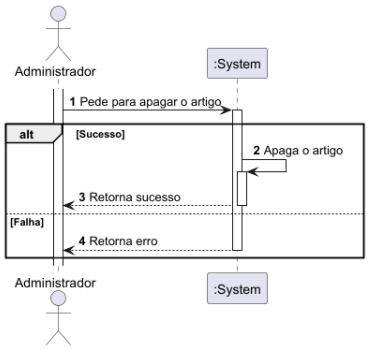


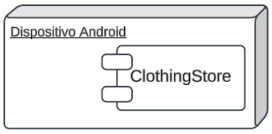
Fig. 11. SSD-UC09 : Apagar artigo

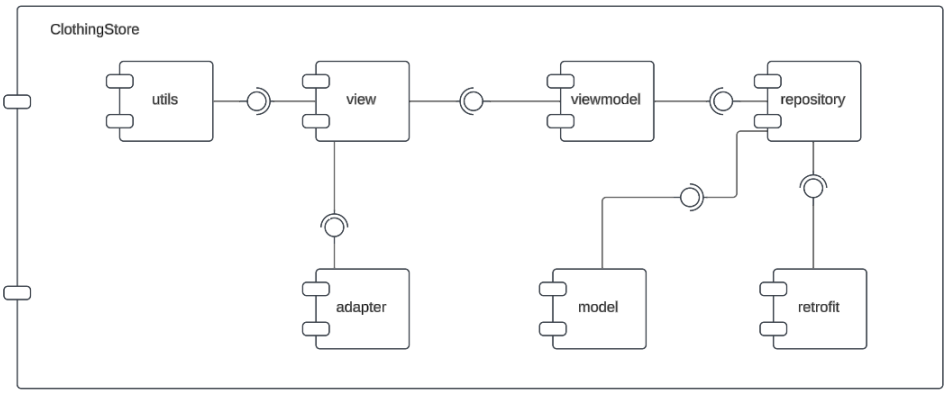
Descrição	Apagar artigo
Pré-condição	Ter sessão de administrador iniciada e ter produto a apagar selecionado

Pós-condição	Artigo apagado
Caminho principal	1. O administrador seleciona a opção para apagar o artigo 2. O sistema apaga o artigo 3. Retorna sucesso
Caminho alternativo	1. Artigo não existe 2. Retorna erro

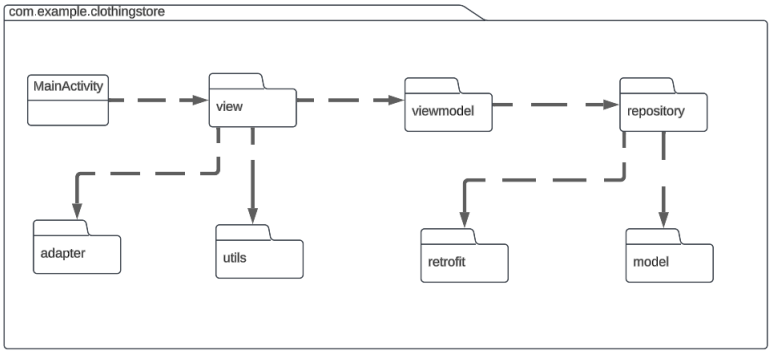
3 Design

3.1 Arquitetura física





3.2 Arquitetura lógica



3.3 Diagrama de classes

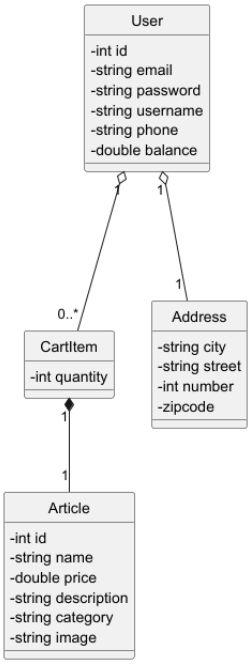


Fig. 12. Diagrama de classes

3.4 Diagramas de sequência

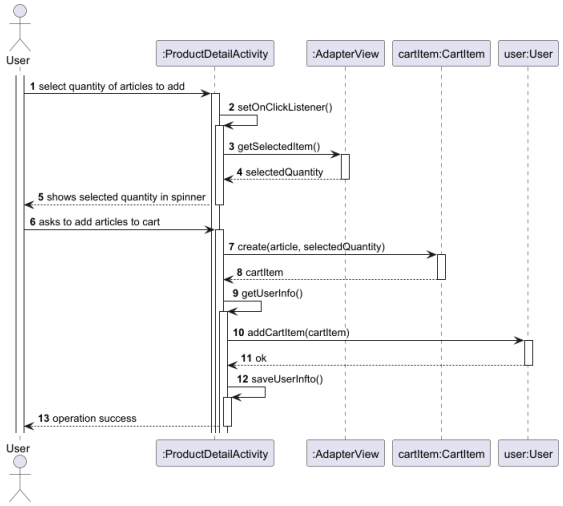


Fig. 13. US01-SD: Adicionar artigo ao carrinho

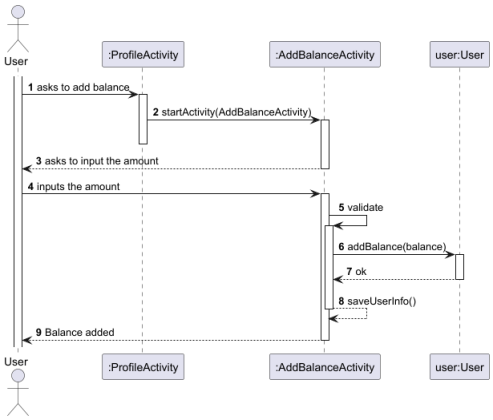


Fig. 14. US02-SD: Adicionar saldo

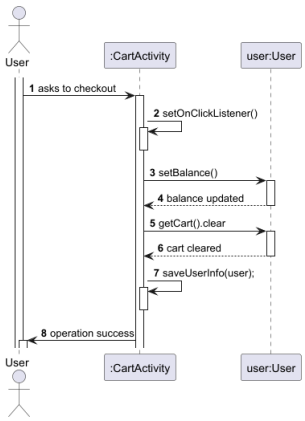


Fig. 15. US03-SD: Checkout

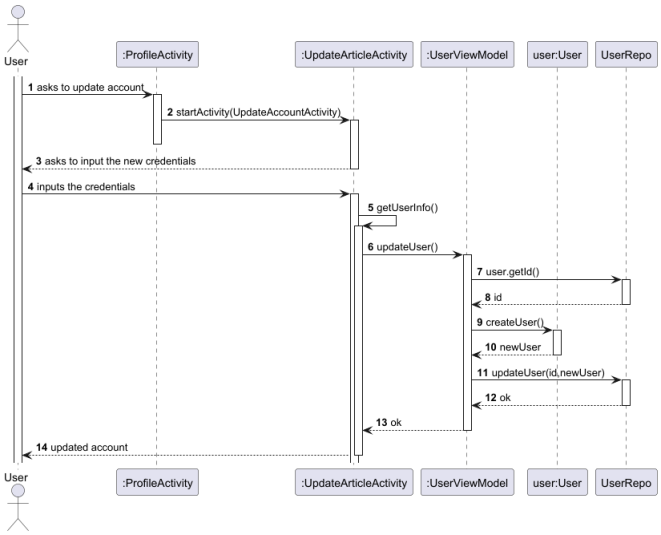


Fig. 16. US04-SD: Atualizar conta de utilizador

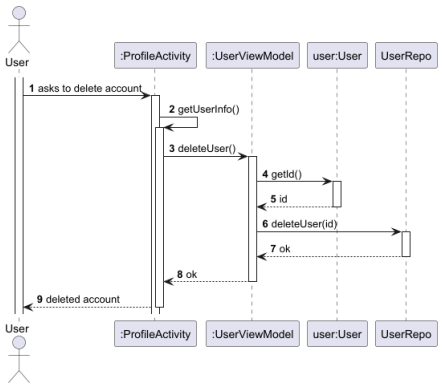


Fig. 17. US05-SD: Apagar conta de utilizador

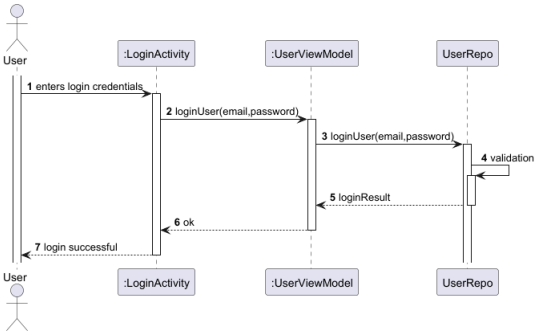


Fig. 18. US06-SD: Login

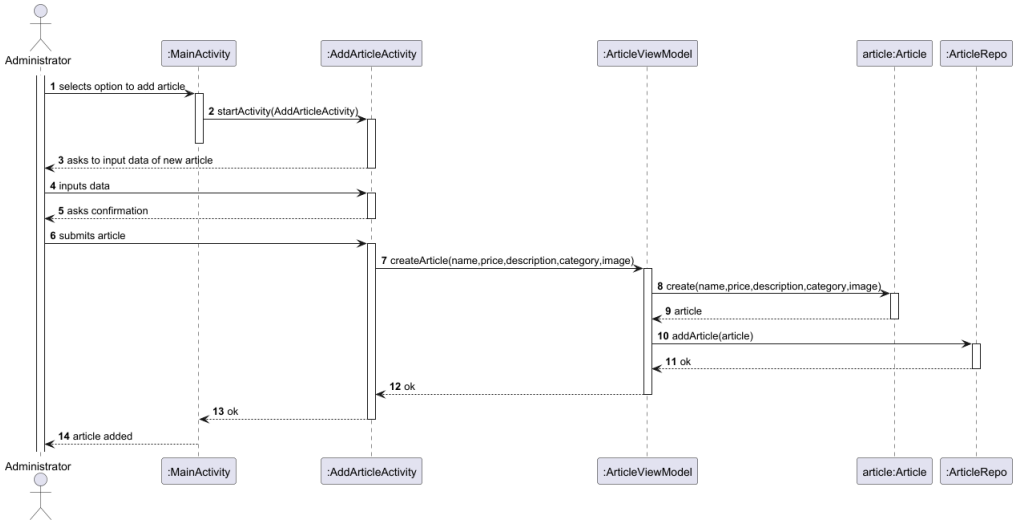


Fig. 19. US07-SD: Adicionar Artigo

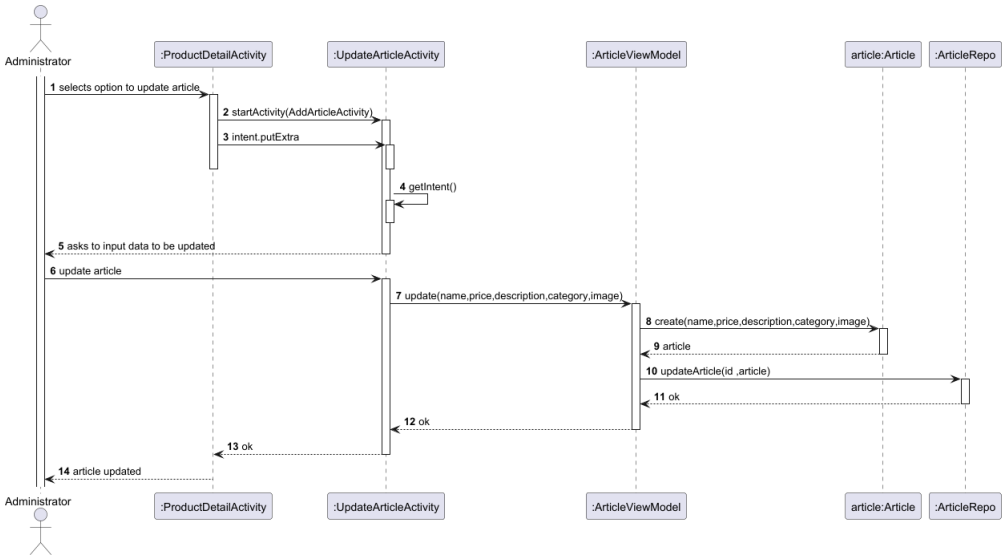


Fig. 20. US08-SD: Atualizar Artigo

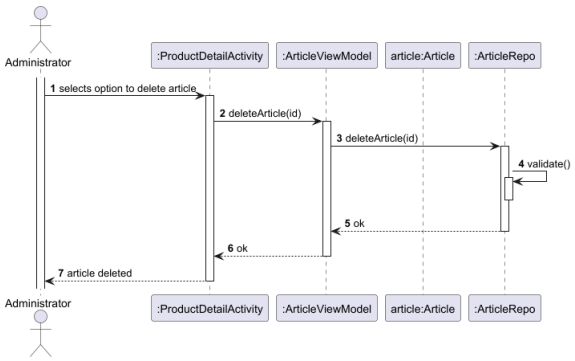


Fig. 21. US09-SD: Apagar Artigo

4 Implementação

– RetroFit

```
retrofit = new Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
    .addCallAdapterFactory(RxJava3CallAdapterFactory.create())
    .build();
```

Retrofit é um cliente HTTP type-safe para Android e Java através do qual realizamos os pedidos request à fake store API. A classe Retrofit.Builder usa a API Builder para permitir a definição do endpoint da URL para as operações HTTP e estabelece que utilizará o Gson como conversor para a deserialização de objetos JSON. É possível observar que utilizamos o RxJava para tratar de operações que devem ser realizadas em uma "background thread".

– Obter artigos

```
public void fetchArticles() {

    Observable<List<Article>> observable = service.getArticles();

    Observer<List<Article>> observer = new Observer<List<Article>>() {
        @Override
        public void onSubscribe(@NonNull Disposable d) {
        }
        @Override
```

```
            public void onNext(@NonNull List<Article> articles) {
                articleList.setValue(articles);
            }
            @Override
            public void onError(@NonNull Throwable e) {
                Log.e("ArticleRepo", e.getMessage());
            }
            @Override
            public void onComplete() {

            }
        };

        observable
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribeWith(observer);
}
```

Este código apresenta o método fetchArticles, utilizado para obter uma lista de artigos da fake store API. Ele cria um Observable para receber os dados e um Observer para tratar os eventos emitidos pelo Observable. As operações são realizadas em uma background thread utilizando Schedulers.io() e os resultados são observados na main thread com AndroidSchedulers.mainThread(). O método trata possíveis erros e atualiza a lista de artigos armazenada no MutableLiveData. Este fluxo assíncrono é ideal para manter a interface responsiva durante operações HTTP.

– Login de utilizadores

```
public MutableLiveData<User> loginUser(String email, String password) {
MutableLiveData<User> loginResult = new MutableLiveData<>();

List<User> currentUserList = userList.getValue();
if (currentUserList != null) {
    for (User user : currentUserList) {
        if (user.getEmail().equals(email)
        && user.getPassword().equals(password))
        {
            loginResult.postValue(user);
            return loginResult;
        }
    }
```

```
        }
        loginResult.postValue(null);
        return loginResult;
    }
```

Este método, loginUser, verifica as credenciais fornecidas (e-mail e password) comparando-as com uma lista de utilizadores armazenada. Se encontrar uma correspondência, retorna os dados do utilizador correspondente encapsulados em um MutableLiveData. Caso contrário, retorna um valor null para indicar que o login falhou. Esta abordagem permite uma comunicação reativa e dinâmica entre os dados do repositório e a interface do utilizador.

– **Utilização de câmera para adição de artigos**

```
        private void openCamera() {
        Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        if (cameraIntent.resolveActivity(getPackageManager()) != null) {
            File photoFile = null;
            try {
                photoFile = createImageFile();
            } catch (IOException e) {
                e.printStackTrace();
                Toast.makeText(this, "Failed to create image file"
                , Toast.LENGTH_SHORT).show();
                return;
            }

            if (photoFile != null) {
                photoUri = FileProvider.getUriForFile(this,
                "com.example.clothingstore.fileprovider", photoFile);

                cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoUri);
                startActivityForResult(cameraIntent, PICK_CAMERA_REQUEST);
            }
        }
    }
```

Este método, openCamera, é responsável por abrir a câmera do dispositivo para capturar imagens que podem ser utilizadas na adição de artigos. Ele cria uma Intent para acionar a ação de captura de imagem e verifica se há suporte no dispositivo. Caso exista, tenta criar um arquivo para armazenar

a foto e, se bem-sucedido, utiliza o FileProvider para gerar um URI seguro. A foto é então armazenada neste URI, e a câmera é iniciada com o método startActivityForResult. Caso ocorra algum erro, uma mensagem é exibida ao utilizador.

5 Testes

– Teste de criação de utilizador

```
    @Test
public void testValidUserCreation() {
    Address address = new Address("Springfield", "12345-5678",
    101, "Main Street");

    User user = new User("test@example.com", "password123", "John Doe",
    "1-234-567-8910", address);

    assertEquals("test@example.com", user.getEmail());
    assertEquals("password123", user.getPassword());
    assertEquals("John Doe", user.getName());
    assertEquals("1-234-567-8910", user.getPhone());
    assertEquals(address, user.getAddress());
}
```

– Teste de criação de artigo

```
    @Test
public void testValidArticleCreation() {

    Article article = new Article("T-Shirt", 25.0f, "A comfortable t-shirt",
    "Clothing", "image.jpg");

    assertEquals("T-Shirt", article.getName());
    assertEquals(25.0f, article.getPrice(), 0.001);
    assertEquals("A comfortable t-shirt", article.getDescription());
    assertEquals("Clothing", article.getCategory());
    assertEquals("image.jpg", article.getImage());
}
```

– **Teste de inserção de password inválida**

```
    @Test
public void testInvalidPassword() {
    Address address = new Address("Springfield", "12345-5678"
    , 101, "Main Street");

    assertThrows(IllegalArgumentException.class, () ->
    new User("test@example.com", null, "John Doe", "1-234-567-8910", address));

    assertThrows(IllegalArgumentException.class, () ->
    new User("test@example.com", "", "John Doe", "1-234-567-8910", address));
}
```

6 Conclusão

Ao longo do desenvolvimento deste projeto, foi adquirido um conhecimento significativo sobre a linguagem Java e o desenvolvimento de aplicações Android. No entanto, a ausência de uma planificação mais rigorosa, bem como de uma análise e design detalhados, tornou-se evidente, especialmente nas etapas finais do projeto. Esta experiência reforça a importância de dedicar mais tempo às fases iniciais de planeamento, garantindo uma base sólida para a implementação e a conclusão do trabalho de forma mais eficiente.