



INEP - APP

Documento de Arquitetura de Software

Enus Nascimento

Versão 1.0.0, 23/08/2023

Índice

1. Introdução	2
1.1. Definições, Acrônimos e Abreviações	2
1.2. Documentos de Referência	2
2. Objetivo do Documento	2
3. Visões da Arquitetura	2
3.1. Critérios de Avaliação Arquitetural	2
3.2. Arquitetura imposta pelo cliente	2
3.3. Arquiteturas descartadas	3
3.4. Visão Geral das Camadas Arquiteturais	3
3.5. Camada de Apresentação	4
3.6. Camada de Negócio (Domain)	4
3.7. Camada de Persistência	5
3.8. Requisitos para Implementação das Camadas	5
3.9. Visão Detalhada das Camadas	5
3.10. Visão Geral do Processo	7
3.11. Visão dos Módulos do Sistema	7
3.12. Rastreabilidade	8
3.13. Visão de Integração	8
3.14. Visão de Classes	8
4. Ambiente de Desenvolvimento	9
4.1. Slidy	9
5. Ambiente de Implantação	10
5.1. Ambiente de Implantação	10
5.2. Componentes Principais	10

Histórico de Revisões

Data	Versão	Descrição	Autor	Revisor
23/08/2023	1.0.0	Criação do documento	Enus Nascimento	Vitor Ferreira

1. Introdução

Este documento visa descrever a arquitetura lógica e física para aplicativos do INEP.

1.1. Definições, Acrônimos e Abreviações

- APP: Faz referência ao Aplicativo
- SSO: O Single Sign-On (Autenticação de Usuários)

1.2. Documentos de Referência

Lista de documentos que foram utilizados como referência na elaboração da arquitetura de software:

- Flutter
- Dart
- Modular
- DotEnv
- Flutter Triple
- Mockito
- Dio
- build_runner

2. Objetivo do Documento

O objetivo deste documento é apresentar ao cliente a descrição da arquitetura do projeto.

3. Visões da Arquitetura

3.1. Critérios de Avaliação Arquitetural

Não se aplica.

3.2. Arquitetura imposta pelo cliente

Não se aplica.

3.3. Arquiteturas descartadas

Não se aplica.

3.4. Visão Geral das Camadas Arquiteturais

Em arquitetura de sistemas, damos o nome de camada a uma estrutura lógica de componentes de um software que interagem entre si para cumprir um objetivo específico, responsabilizando-se pela execução de uma dada tarefa ou um conjunto de tarefas semelhantes. Em alguns casos, esse agrupamento pode não ser apenas lógico, mas sim assumir um caráter físico. Isso ocorre quando uma camada possui a habilidade de ser executada separadamente do restante da aplicação.

A arquitetura baseada em múltiplas camadas oferece um modelo flexível e reutilizável para o desenvolvimento de software. Ao se dividir uma aplicação em camadas, evitamos o forte acoplamento entre os componentes, de forma que a maioria das alterações necessárias possam ser feitas de maneira mais isolada e pontual, reduzindo o esforço necessário à manutenção do sistema.

Na figura abaixo, apresentamos uma visão simplificada das camadas que compõem o sistema e a forma como elas se relacionam. Nas seções a seguir, detalhamos conceitualmente cada camada e enumeramos as ferramentas e plataformas que serão utilizadas para sua construção.

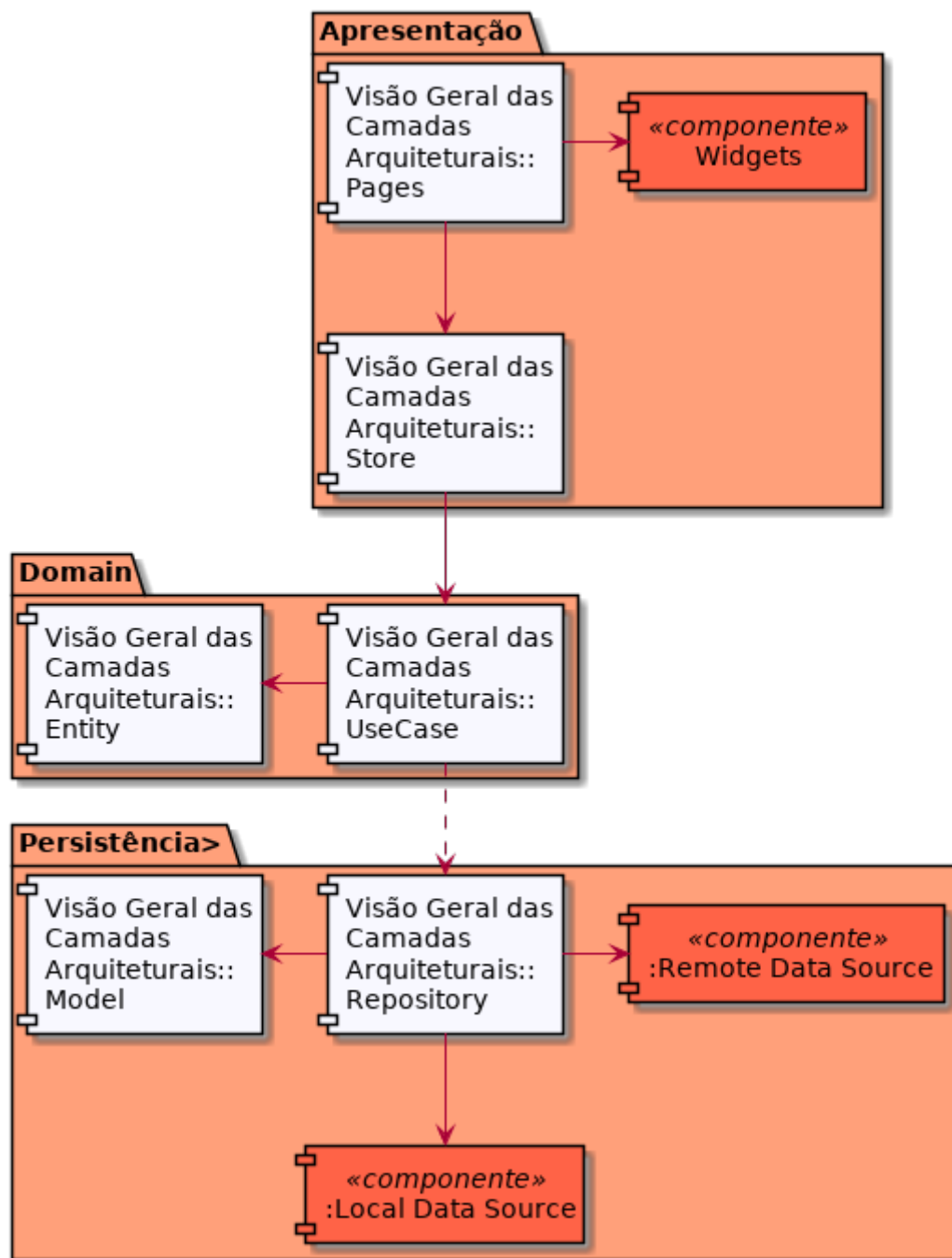


Imagem 1. Visão Geral das Camadas Arquiteturais Aplicação Móvel

3.5. Camada de Apresentação

A camada de apresentação é responsável por fazer a lógica de construção das páginas para serem exibidas pelos usuários, tratar os eventos do browser, como cliques, e gerenciar o fluxo de execução do sistema.

3.6. Camada de Negócio (Domain)

A camada de negócios é responsável pela implementação lógica da aplicação. Ela expõe

os serviços para a camada de apresentação por meio de uma interface bem definida e obtém as informações necessárias para mostrar ao usuário por meio da Camada de Persistência.

3.7. Camada de Persistência

A camada de persistência é responsável pela lógica de acesso ao banco de dados e pelo mapeamento dos dados em entidades representativas. O objetivo em mapear o banco de dados em entidades representativas ao sistema é diminuir a diferença semântica entre o modelo abstrato do banco e o problema do mundo real.

3.8. Requisitos para Implementação das Camadas

Os itens apresentados a seguir referem-se à lista de tecnologias e requisitos necessários para suportar as camadas de apresentação, negócio e persistência da aplicação móvel:

- Flutter ~3.13.0
- Dart ~3.1
- Dio ^5.0
- flutter_modular ^6.0.0
- flutter_triple ^3.0.0
- flutter_dotenv ^5.1.0

Bibliotecas a serem utilizadas no ambiente de desenvolvimento para a aplicação móvel:

- mockito: ^5.4.0
- flutter_lints: ^2.0.0
- triple_test: ^2.0.0
- modular_test: ^2.0.0
- http_mock_adapter: ^0.5.0
- build_runner: ^2.4.6

3.9. Visão Detalhada das Camadas

A seguir, apresentamos uma visão detalhada das camadas do sistema, na qual são ilustrados os principais componentes que compõem cada camada bem como os sistemas externos com o qual o sistema deverá interagir.

3.9.1. Aplicação móvel

Esta camada é formada basicamente por componentes do tipo Page, Widget e o Gerenciamento de estados (Store). O pacote **flutter_triple** foi adotado para auxiliar no gerenciamento de estado. Esse pacote segmenta o estado em 3 partes reativas, o valor do estado (state), o objeto de erro (error) e a ação de carregamento (loading). Esses seguimentos podem ser observados de forma individual como no exemplo a seguir:

```
counter.observer(  
  onState: (state) => print(state),  
  onError: (error) => print(error),  
  onLoading: (loading) => print(loading),  
);
```

A camada de negócio é composta por Entidades e UseCases. As Entidades são estruturas simples que representam a estrutura de dados do negócio. Os UseCases representam as funcionalidades do aplicativo.

O pacote DartZ é utilizado para auxiliar no tratamento de erros nessa camada. Esse pacote adiciona suporte a um tipo `Both<Left, Right>`, que utilizaremos para verificar se o resultado de uma operação ocorreu com sucesso - Right, ou se ocorreu uma falha - Left. Essa abordagem impõe a implementação de tratamento de erros, evitando falhas inesperadas no tempo de execução. O método `fold` pode ser utilizado para verificar ambos os valores como no exemplo a seguir:

```
Future<Either<Failure, List<Noticia>>> call(String query) async {  
  var option = optionOf(query);  
  
  return option.fold(  
    () => Left(InvalidText()),  
    (text) async {  
      var result = await repository.search(text);  
  
      return result.where((r) => r.isNotEmpty, () => EmptyList());  
    },  
  );  
}
```

Essa camada é composta por componentes do tipo Repository, Models e DataSource. Aqui utilizaremos os recursos **left** e **right** do pacote DartZ que através de um tratamento de errors, será possível retorna uma resposta diferente de acordo com o resultado obtido. O exemplo abaixo ilustra essa abordagem:


```

Future<Either<Failure, List<NoticiaModel>>> search(String query) async {
    List<NoticiaModel> resultList;

    try {
        resultList = await datasource.search(query);
    } catch (e) {
        return left(InvalidText());
    }

    return resultList == null ? left(ActionResult()) : right(resultList);
}

```

3.10. Visão Geral do Processo

O diagrama de sequência abaixo apresenta o fluxo de informações do sistema e suas interfaces através das camadas arquiteturais.

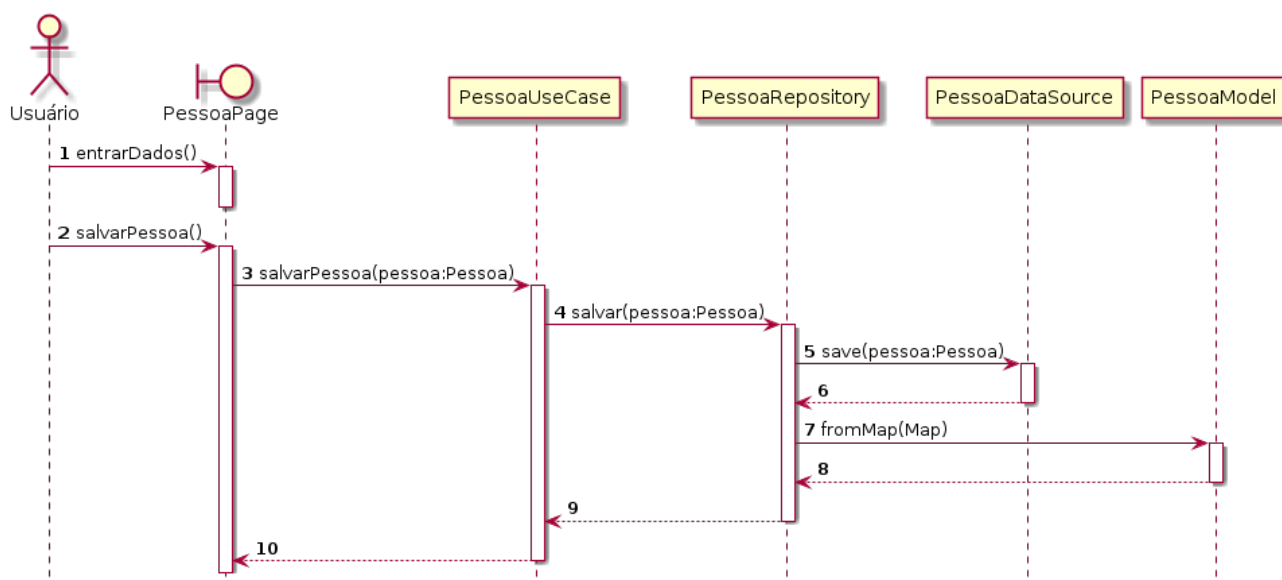


Imagem 2. Visões da arquitetura

3.11. Visão dos Módulos do Sistema

Neste item apresentamos uma visão geral dos módulos do projeto, ilustrando-os graficamente através de um diagrama. Em seguida, detalhamos as responsabilidades de cada módulo, em concordância com a especificação funcional.

Serão criados os seguintes módulos:

- **SSO:** Autenticação de usuários;
- **Admin:** Módulo que contém as funcionalidades protegidas (O usuário de estar autenticado para ter acesso a essas funcionalidades);

- **Public** Módulo para apresentação de informações sem a necessidade de autenticação.

3.12. Rastreabilidade

Nesta seção, é apresentada a matriz de rastreabilidade no nível de componentes arquiteturais do produto, o que permite rastrear os fontes que apoiam a realização dos requisitos propostos para o sistema alvo.

Origem do Requisito	Item no DAS
OS: Definição das tecnologias	3.10. Requisitos para Implementação das Camadas

3.13. Visão de Integração

A visão de integração apresenta as integrações pertinentes ao sistema. Essas integrações podem ser internas ou externas e é de suma importância que os projetistas e desenvolvedores conheçam em detalhes essas integrações, bem como os contratos e protocolos de comunicação entre os sistemas/componentes envolvidos na integração.

O Sistema deve se integrar aos seguintes sistemas a fim de atender os requisitos funcionais e não-funcionais da aplicação:

- SSO

3.13.1. SSO

O Single Sign-On (SSO) é um método de autenticação e autorização que permite que usuários acessem múltiplas aplicações com apenas um único login — nome de usuário e senha, por exemplo. O projeto fará integração com o SSO por meio do protocolo **OpenID**.

3.14. Visão de Classes

Nesta seção, estão descritos os diagramas/fluxogramas de classe, sequencia e atividades para os requisitos críticos do sistema.

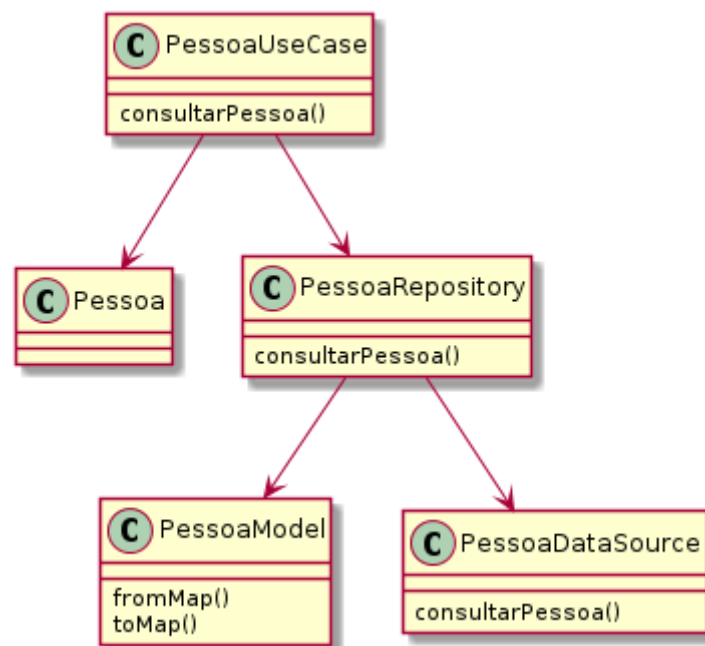


Imagem 3. Diagrama de classes

4. Ambiente de Desenvolvimento

Ferramentas a serem utilizadas no ambiente de desenvolvimento:

- Visual Studio Code
- Android Studio
- Flutter ~3.13.0
- Dart ~3.1
- Slidy 4.0.4

Ferramentas necessárias para dar suporte à aplicação e ao desenvolvimento são:

- DBeaver

A solução prevê acessos às seguintes bases de dados:

- SQLite

4.1. Slidy

A ferramenta Slidy foi adotada para ajudar a estruturar o projeto de forma padronizada, organizando o aplicativo em Módulos formados por páginas, repositórios, widgets, triplo, além de criar testes unitários automaticamente. O Módulo oferece uma maneira mais fácil de injetar dependências e blocos, incluindo descarte automático. Maiores informações a

respeito da sua utilização podem ser encontradas na [Documentação do Projeto](#)

5. Ambiente de Implantação

5.1. Ambiente de Implantação

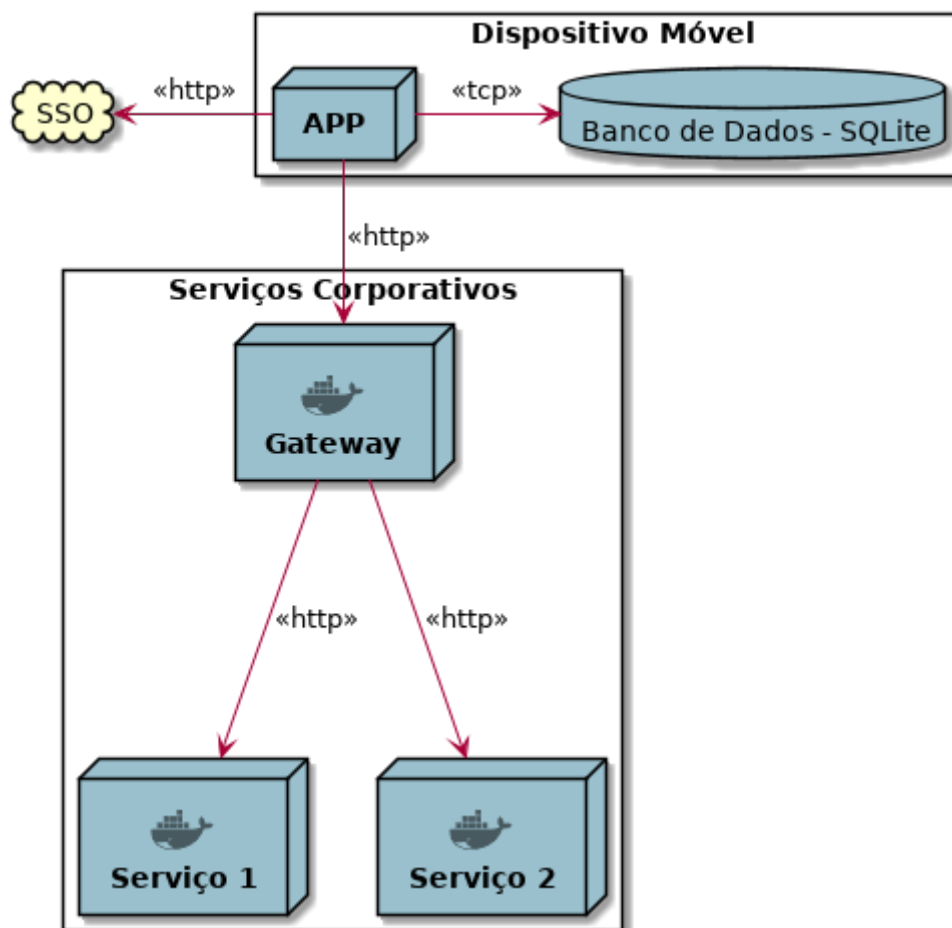


Imagem 4. Diagrama de Implantação

5.2. Componentes Principais

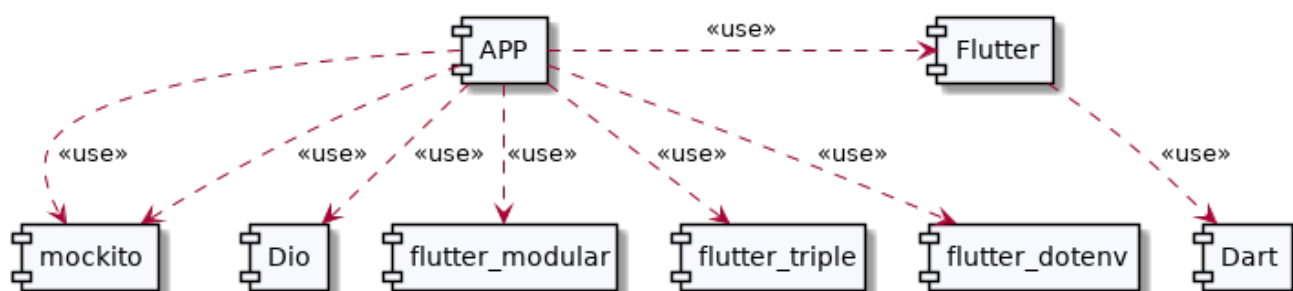


Imagem 5. Componentes Aplicação Móvel