MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

PROGRAMA DE RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO
TRIBUNAL DE CONTAS DO ESTADO DO RIO GRANDE DO NORTE

CONSTRUÇÃO DOS MÓDULOS DE NOTÍCIAS E PROCESSOS DA APLICAÇÃO MOBILE DO TCE-RN

YLLAN VINÍCIUS FERREIRA GURGEL

Trabalho de Conclusão de Curso apresentado ao Programa de Residência em Tecnologia da Informação da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Residente em Tecnologia da Informação.

Orientador
Itamir de Morais Barroca Filho

Natal, RN
Junho de 2022

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Gurgel, Yllan Vinicius Ferreira.
Construção dos módulos de notícias e processos da aplicação mobile do TCE-RN / Yllan Vinicius Ferreira Gurgel. - 2022.
34 f.: il.

Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Residência em Tecnologia da Informação.
Orientador: Prof. Dr. Itamir de Morais Barroca Filho.

1. Mobile - Dissertação. 2. Flutter - Dissertação. 3. Aplicativo - Dissertação. I. Barroca Filho, Itamir de Morais. II. Título.

RN/UF/BCZM CDU 004.7

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO

Aluno	Yllan Vinícius F. Gurgel
Matrícula	
Data da Defesa	24/06/2022
Hora da Defesa	10:30
Local da Defesa	Google Meet (vídeo-conferência)
Título do Trabalho	Construção dos módulos de notícias e processos da aplicação mobile do TCE-RN

Examinador (Nome e Afiliação)	**Nota**	**Rubrica**
ORIENTADOR: Itamir de Morais Barroca Filho	8,0	
Marcel Vinicius Medeiros Oliveira	7,5	MARCEL VINICIUS MEDEIROS OLIVEIRA:02386943488 Assinado de forma digital por MARCEL VINICIUS MEDEIROS OLIVEIRA:02386943488 Dados: 2022.06.28 15:35:02 -03'00'
Lindemberg Silva Pereira	9,0	
MÉDIA FINAL	8,2	

FAIXA DE MÉDIA FINAL	CONCEITO FINAL	STATUS
0,0 a 2,9	E	REPROVADO
3,0 a 4,9	D	REPROVADO
5,0 a 6,9	C	APROVADO
7,0 a 8,9	B	APROVADO
9,0 a 10	A	APROVADO

TERMO DE APROVAÇÃO DE
VERSÃO FINAL DO TRABALHO DE CONCLUSÃO DE CURSO

Atesto que o referido Trabalho de Conclusão de Curso foi considerado aprovado pela Banca Examinadora e que todas as solicitações de correção feitas por esta Banca foram atendidas satisfatoriamente dentro do prazo, ficando o aluno, portanto, com **CONCEITO FINAL** B na atividade.

Natal, 24 de junho de 2022.

_____
ORIENTADOR

Construção dos módulos de notícias e processos da aplicação mobile do TCE-RN

Yllan Vinícius F. Gurgel[1], Itamir de Morais B. Filho[2]

[1,2]Instituto Metrópole Digital (IMD) – Universidade Federal do Rio Grande do Norte (UFRN) Avenida Capitão Mor Gouveia, 3000, 59078-970, Natal – RN – Brasil

the_yllan@hotmail.com, itamir.filho@imd.ufrn.br

Abstract. *The use of mobile devices to solve day-to-day problems is a reality that we cannot ignore these days. The TCE (Tribunal de Contas do Estado do Rio Grande do Norte), like most public institutions, is also concerned with bringing its services to users in an easy way through applications for mobile devices. Currently, the court depends on a third-party company for the development and maintenance of the mobile application, which means that it has less control over its general functioning, making maintenance expensive and bureaucratic. This work aims to describe the development process of the Processes and News modules of the TCE application, approaching the choice of hybrid technology (Flutter), the advantages and disadvantages of this approach, choice of project architecture, development, difficulties encountered and improvements to be considered for the continuity of the project.*

Resumo. *A utilização de dispositivos móveis para resolver problemas do dia a dia é uma realidade que não podemos ignorar nos dias atuais. O TCE (Tribunal de Contas do Estado do Rio Grande do Norte), assim como a maioria das instituições públicas, também tem essa preocupação de levar seus serviços de forma facilitada para os usuários através de aplicações para os dispositivos móveis. Atualmente o tribunal depende de uma empresa de terceiros para desenvolvimento e manutenção do aplicativo móvel, o que faz com o que ela tenha menos controle sobre o funcionamento geral do mesmo, tornando cara e burocrática a manutenção. Esse trabalho tem como objetivo descrever o processo de desenvolvimento dos módulos de Processos e Notícias do aplicativo do TCE , abordando a escolha da tecnologia híbrida (Flutter), as vantagens e desvantagens dessa abordagem, escolha de arquitetura do projeto, desenvolvimento, dificuldades encontradas e melhorias a serem consideradas para a continuidade do projeto.*

1. Introdução

O TCE/RN (Tribunal de Contas do Rio Grande do Norte) tem como papel principal a fiscalização sobre os gastos realizados pelos órgãos subordinados ao mesmo, sendo eles: prefeituras, câmaras, governo do estado, fundos e autarquias. Por serem órgãos de domínio público e utilizarem de verba pública em geral através de diversos recursos como

impostos, a transparência e confiabilidade dos trabalhos realizados e dos dados obtidos nas mais diversas análises dos gastos públicos são de suma importância para uma sociedade mais informada, politicamente inclusa e mais organizada. Através desses princípios pode se ter uma maior segurança de que o dinheiro que estamos investindo através do nosso trabalho está sendo aplicado de forma correta em recursos para nossa qualidade de vida em geral.

Dada a importância da transparência dos dados com os quais o tribunal trabalha em seu dia a dia, é importante também se pensar em disponibilizar os mesmos de maneira a facilitar e aumentar o alcance da população a esses dados. E quando falamos de acesso a dados, a importância de levarmos isso para o escopo dos dispositivos móveis está intrínseca nessa análise.

Tendo isso em mente, o TCE/RN disponibiliza hoje em dia um aplicativo chamado TCE RN Digital, onde a população como um todo tem acesso à uma série de serviços relacionados à consulta processual, notícias, escola de contas, ouvidoria, pautas de sessões, painel de obras e painel da transparência.

Apesar da gama de serviços oferecidos através do aplicativo, existem alguns problemas na maneira como isso foi pensado e disponibilizado. Hoje em dia o tribunal terceiriza a construção e manutenção desse aplicativo a uma empresa externa ao órgão. Essa abordagem traz uma série de questões que devem ser levadas em consideração tais como: maior dificuldade de comunicação (já que a empresa é externa ao tribunal), maior burocracia na hora de analisar e sugerir alterações e/ou novas funcionalidades ao aplicativo, além de ter um custo alto (não divulgado pelo tribunal) para a manutenção da aplicação como um todo.

Com essa problemática em mente, foi pensado em trazer a responsabilidade da construção e manutenção desse aplicativo para o tribunal, facilitando assim a continuidade da disponibilização desses serviços para a população atendendo as demandas de forma cada vez mais rápida, possibilitando um sistema mais flexível para mudanças e escalabilidade, além de diminuir o custo do tribunal com esse recurso.

Esse artigo tem como objetivo geral abordar o processo de desenvolvimento dos módulos de Notícias e Processos da aplicação mobile TCE RN Digital, mostrando as decisões tomadas, o embasamento teórico utilizado para isso, as ferramentas utilizadas, as dificuldades encontradas no processo e os próximos passos para a continuidade do projeto.

2. Entendimento da Demanda

Para iniciarmos o desenvolvimento da aplicação, procurou-se seguir alguns passos que trariam um melhor entendimento da problemática possibilitando escolhas mais assertivas acerca dos métodos, tecnologias utilizadas, decisões arquiteturais da solução e de design de código.

Primeiramente foi realizada uma entrevista com o diretor da DIN (Diretoria de Informática) para um melhor entendimento do estado atual da aplicação. Como dito anteriormente, será abordado o entendimento relativo aos módulos de Notícias e Processos.

2.1 Módulo de Notícias

Atualmente o módulo de notícias no aplicativo do TCE consiste apenas em exibir as últimas notícias divulgadas na página inicial da aplicação, a exibição da listagem de todas as notícias ativas e uma página para exibir os detalhes da notícia.

Para a obtenção dessas notícias, não existe uma forma de integração construída especificamente para isso. O aplicativo se utiliza da técnica de *web scrapping*, que segundo (HERNÁNDEZ et al., 2015) "é o processo de rastreamento e download de sites de informações e extração de dados não estruturados ou mal estruturados em um formato estruturado", sendo o site o qual se faz esse processo a área restrita do tribunal(https://novaarearestrita.tce.rn.gov.br). Apesar de funcional, essa não é a melhor forma de fornecer esses dados. Qualquer mudança que ocorra no site da área restrita pode impactar na obtenção dessas informações pela aplicação móvel, esse forte acoplamento entre os sistemas dificulta uma possível evolução em ambos, e isso se agrava pelo fato de que mudanças frequentes na aplicação móvel gera mais custo para o tribunal tendo em vista a terceirização da manutenção.

2.2 Módulo de Processos

O módulo de Processos ou Consulta Processual presente no aplicativo TCE RN Digital tem como objetivo dispor de dados referentes aos processos que correm dentro do tribunal. O usuário do sistema pode obter várias informações de processos informando alguns dados utilizando-se de um formulário disponibilizado no sistema.

Feita a busca, logo abaixo do formulário são mostrados os resultados obtidos de acordo com os filtros fornecidos em formato de tabela.

É possível visualizar o detalhamento do processo, tais como pessoas associadas, movimentações, apensos e comunicações. Porém essa funcionalidade nos redireciona para o próprio site do TCE, atualmente saindo do escopo da aplicação.

2.3 Requisitos do Projeto

Ao fim dessa análise do estado atual da aplicação foram passados alguns requisitos que têm uma maior prioridade nesse momento inicial da migração desse projeto para algo interno do tribunal.

No módulo de notícias, foi informada a necessidade de uma melhor maneira da obtenção dos dados, que diminuísse o acoplamento do site da Área Restrita e o aplicativo TCE RN Digital. O layout da aplicação que diz respeito a esse módulo seria mantido em um primeiro momento.

No módulo de processos, a necessidade informada além da construção das telas já presentes na aplicação, foi trazer o caso de uso de visualização dos detalhes do processo para dentro da aplicação. Essa alteração faz com que a experiência do usuário seja mais consistente, tendo ele todas as informações que precisa sem sair do escopo do aplicativo. Além disso, um modo de trazer uma melhor visualização dos resultados das consultas também foi solicitado, já que atualmente a funcionalidade não agrada os *stakeholders*, conforme dito em entrevistas realizadas com os mesmos.

3. Desenvolvendo a aplicação

Dando início ao desenvolvimento, o primeiro passo a se pensar foi a arquitetura da solução. Arquitetura de software é um conceito até hoje muito discutido e que é difícil se chegar a uma definição única. Dos muitos conceitos apresentados em diversas literaturas, um dos que melhor expressa o entendimento para a realização desse trabalho é a definição de (BASS et al., 2021), "a arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles". Tendo essa ideia em mente chegou-se ao diagrama C4 da **Figura *1*** que representa a solução descrita neste artigo:

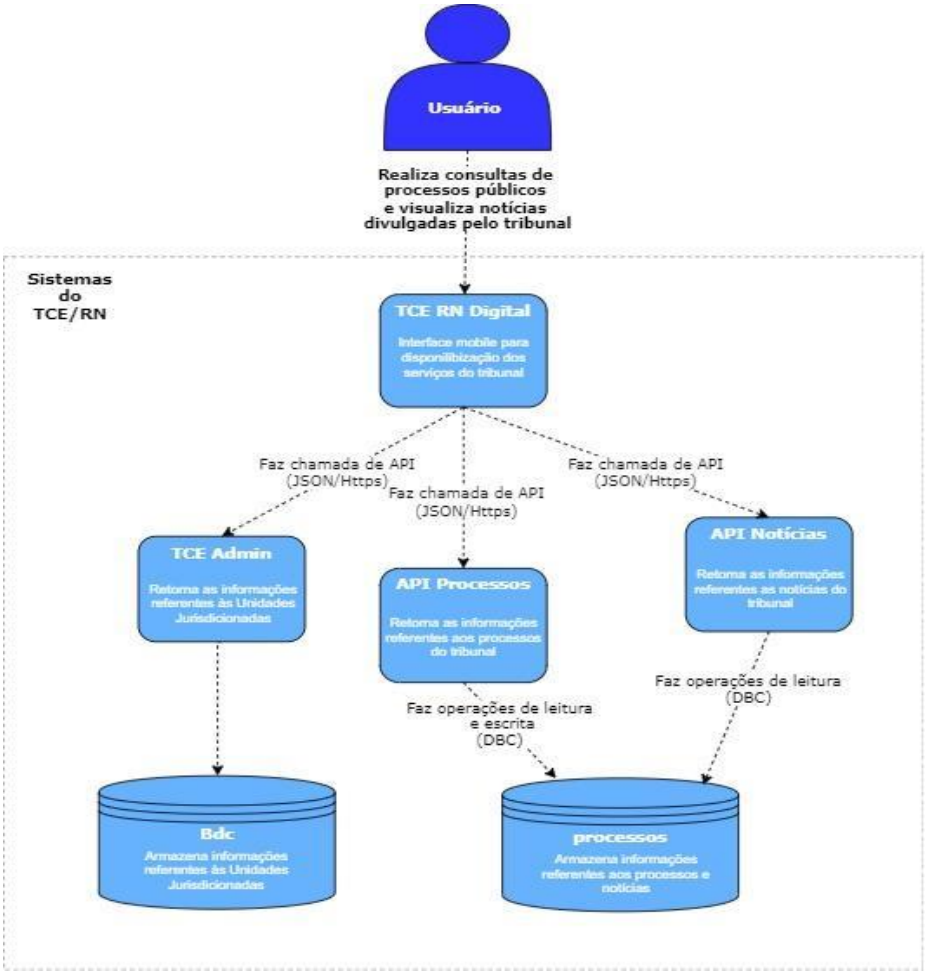

Figura 1. Diagrama C4 (nível 2) da arquitetura da solução

Outra questão muito importante a ser abordada sobre decisões arquiteturais é que não existe bala de prata quando falamos desse assunto. Assim como a arte, a arquitetura só pode ser entendida dentro de um contexto. Muitas decisões arquiteturais são feitas sempre se baseando na realidade do ambiente em que aquela solução está presente [Richards e Ford 2020]. Será detalhado a seguir cada um dos componentes mostrados acima.

3.1 Aplicação Mobile

Quando falamos de desenvolvimento mobile, nos deparamos logo com uma decisão importante: Nativo *vs* Híbrido.

Ao se falar em desenvolvimento nativo estamos nos referindo a aplicativos que são desenvolvidos para um tipo específico de plataforma [Silva e Santos 2014]. Hoje em dia quando falamos de plataformas mobile nos deparamos em dois sistemas dominantes, Android e iOS. Para cada uma dessas plataformas são utilizadas diferentes linguagens de programação (Kotlin/Java para Android e Swift para iOS) e ambientes de desenvolvimento. Portanto, desenvolver aplicativos nativos para esses dois sistemas operacionais separadamente requer uma equipe de trabalho com conhecimento nessas diversas tecnologias [Charland e Leroux 2011 apud Silva e Santos 2014].

Algumas vantagens da utilização dessa abordagem são: a ótima experiência do usuário devido ao acesso facilitado aos recursos do celular (câmera, GPS, entre outros), os componentes de um aplicativo nativo são os mesmos do sistema operacional tornando a utilização do aplicativo mais intuitiva ao usuário e a independência de comunidades responsáveis por versões de frameworks de desenvolvimento já que as atualizações das ferramentas de desenvolvimento vêm em conjunto com as atualizações do próprio sistema operacional alvo [Silva e Santos 2014].

Apesar de todas as vantagens, a necessidade de desenvolvedores com especialidades diferentes para atender a todas as plataformas foi fator decisivo para a não adoção do desenvolvimento nativo dado o contexto atual do tribunal.

Por outro lado, o desenvolvimento híbrido ou *cross-platform*, refere-se ao processo de desenvolvimento para múltiplas plataformas com um único código a ser compilado sem a necessidade de reescrevê-lo, tornando estas aplicações mais práticas e econômicas para o mercado [Neves e Junior 2020]. *Flutter* e *React Native* são os frameworks mais utilizados atualmente.

De acordo com Palmieri et al. (2012) essa abordagem trás benefícios como: redução da complexidade, redução de código, redução do tempo de desenvolvimento e custo de manutenção, diminuição de conhecimentos necessários sobre a API nativa, maior facilidade no desenvolvimento e aumento de participação de mercado. A principal desvantagem está relacionada a dependência da comunidade para atualização das ferramentas de desenvolvimento conforme os sistemas operacionais sofrem atualizações e possíveis gargalos de performance relacionados à conversão dos componentes do framework para o nativo.

Dado o contexto atual do tribunal, a decisão foi a utilização do desenvolvimento híbrido com *Flutter* e, portanto, será dessa ferramenta que falaremos mais a fundo neste trabalho.

3.1.1 Flutter

Flutter é um *toolkit* de UI (*User Interface*) desenvolvido para permitir reuso de código entre diferentes sistemas operacionais tais como iOS e Android, permitindo também que as aplicações desenvolvidas interajam diretamente com os serviços internos dessas plataformas [Flutter 2022].

Esse framework nos permite desenvolver interfaces de usuário de maneira *"declarativa"*, ou seja, a interface do usuário nunca será chamada de forma *"imperativa"* ou explícita no código. Ao invés disso, o código declara que a interface deve ser mostrada de certa maneira, dado um certo estado. Outra maneira de se pensar sobre esse conceito é imaginar que toda a interface do usuário de uma aplicação Flutter é o resultado de uma função que recebe o estado do aplicativo em execução como parâmetro [FAUST 2020].

Para ilustrar o funcionamento do flutter, e o porquê de uma adoção cada vez maior por ele no mercado temos a **Figura *2***:

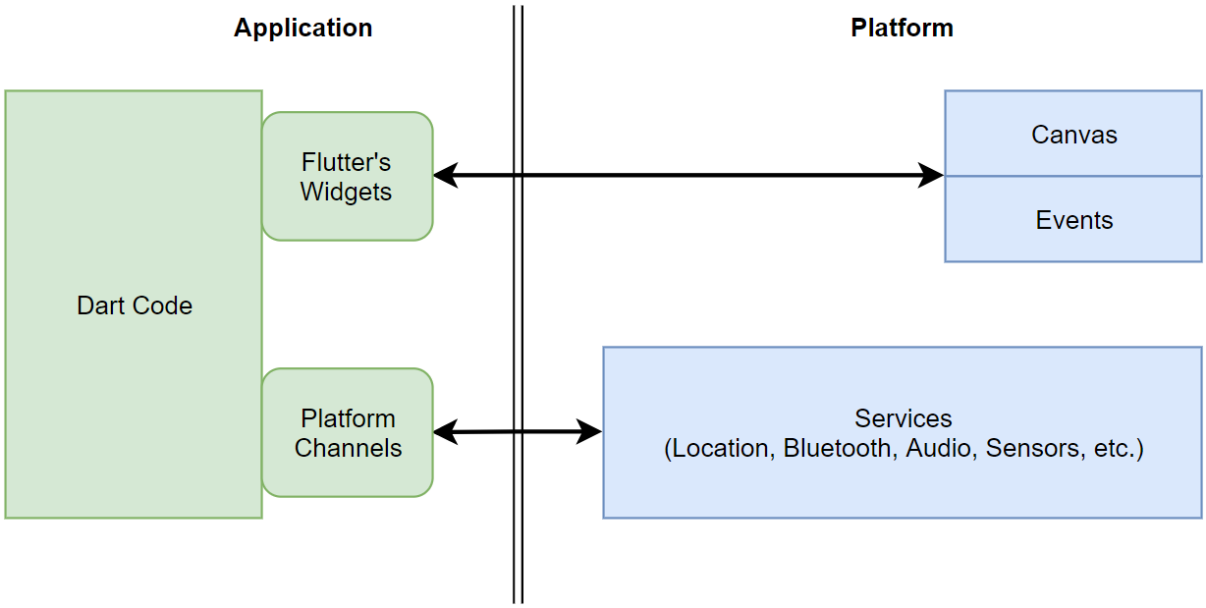

Figura 2. Diagrama de referência do funcionamento do Flutter (FAUST, 2020)

A abordagem do Flutter é mover todo o processo de renderização para dentro da aplicação. A renderização é feita com a própria *engine* do Flutter e seus próprios *widgets* ou componentes. Esse framework se comunica diretamente com o canvas da plataforma em que está rodando. Essa abordagem reduz o gargalo que poderia ser causado por *bridging*. A comunicação entre a aplicação Flutter e os serviços nativos da plataforma são feitos através de *Platform Channels* [FAUST 2020].

Escolhido o framework, passamos à escolha de qual arquitetura utilizar dentro dessa aplicação sempre pensando em uma maior flexibilidade, manutenibilidade e o escopo atual da aplicação. Quando pensamos em arquitetura mobile temos alguns padrões mais conhecidos como MVC, MVVM e MVP [Daoudi et al 2019]. Nesse artigo abordaremos o padrão MVC, o qual foi utilizado para o desenvolvimento da aplicação mobile.

3.1.2 Padrão Arquitetural MVC

Proposto em 1979, MVC tem como objetivo separar a lógica de negócio da camada de apresentação. É definido por três camadas principais. Primeiro, o *Model*: basicamente um

conjunto de entidades que representam o conhecimento, informação e o conjunto de regras sobre atualizações e acesso a essas informações. Segundo, a *View*, que é encarregado de prover a representação visual do *Model*. Terceiro, o *Controller,* que é o ponto de entrada da aplicação, manipula a *View* e traduz a interação do usuário com o *Model* [Daoudi et al 2019].

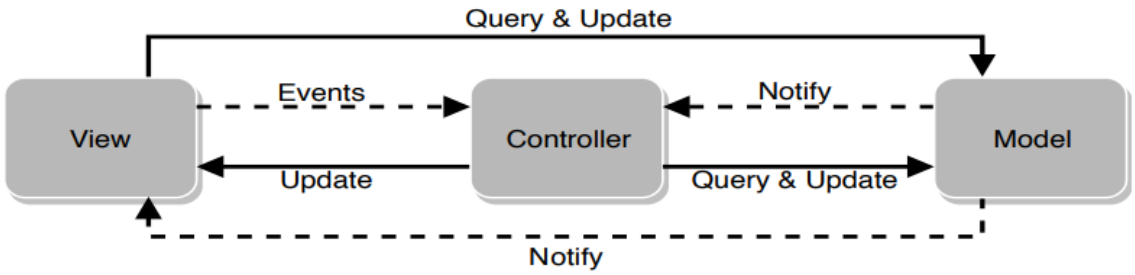

Figura 3. Esquema do MVC Clássico (DAOUDI et al, 2019)

3.1.2 Modularização da aplicação e boas práticas

Pensando em melhorar mais ainda a manutenção e legibilidade do código do projeto, foi realizada a modularização da aplicação, dividindo os escopos de maneira física e lógica, trazendo assim uma melhor organização para o projeto, uma maior independência entre esses módulos da aplicação e explicitando os limites arquiteturais a serem seguidos. Essa abordagem é utilizada de maneira padrão no Angular, outro framework web, que é utilizado na *stack* de desenvolvimento web para interfaces de usuário do TCE/RN.

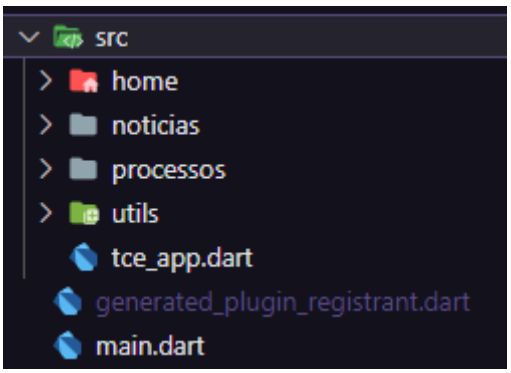

Figura 4. Estrutura de pastas do projeto Flutter

A pasta *home* diz respeito ao módulo da parte de inicialização do aplicativo, a pasta *noticias* tem todos o escopo do módulo de Notícias, a pasta de *processos* contendo o escopo de Processos. Cada um desses módulos têm a separação de responsabilidades indicado no padrão MVC. A pasta *utils* consiste em componentes que são compartilhados por toda a aplicação.

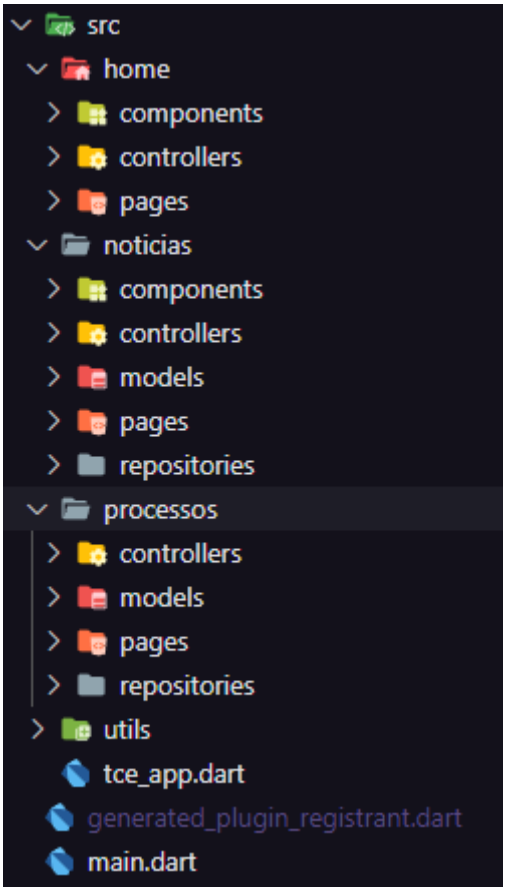

Figura 5. Estrutura interna de pastas dos módulos da aplicação

Algumas adaptações foram feitas nessa estruturação. A pasta *pages* representa a camada de *view,* mudando somente a nomenclatura de modo a expor que os arquivos ali contidos dizem respeito a telas inteiras da aplicação. A pasta *components* também faz parte da camada de *view*, porém com *widgets*/componentes menores e adaptados para a aplicação. Essa abordagem é utilizada quando se tem um elemento de interface visual que se repete muitas vezes na tela, e essa separação em componentes menores facilita a reutilização dos mesmos para construção das telas (*pages*), agilizando o desenvolvimento.

A pasta *models*, como o próprio nome já diz, contém as entidades do escopo da nossa aplicação. No nosso caso seriam as Notícias e os Processos, além de outras entidades auxiliares como Relatores e Órgãos, que estão envolvidos na regra de negócio para obtenção dos dados das entidades principais.

```dart
class Processo {
  final int idProcesso;
  final String numero;
  final String interessado;
  final String assunto;
  final String tipo;

  Processo({
    required this.idProcesso,
    required this.numero,
    required this.interessado,
    required this.assunto,
    required this.tipo,
  });

  static Processo fromJson(Map json) {
    return Processo(
      idProcesso: json["idProcesso"] ?? 0,
      numero: json["numeroProcesso"] ?? "",
      interessado: json["interessado"] ?? "",
      assunto: json["assunto"] ?? "",
      tipo: json["tipoNome"] ?? "",
    );
  }
}
```

Figura 6. Classe representando o Model de Processo

A pasta *controllers* também corresponde ao conceito abordado no padrão arquitetural do MVC. No projeto desenvolvido foi colocado um *controller* referente a cada tela, cada um contendo a regra de negócio associada à respectiva tela.

```dart
class ProcessosController extends ChangeNotifier {
  final IOrgaosRepository _orgaosRepository;
  final IRelatoresRepository _relatoresRepository;
  final IProcessosRepository _processosRepository;

  ProcessosController(
    this._orgaosRepository,
    this._relatoresRepository,
    this._processosRepository,
  ) {
    _loadResources();
  }

  String numeroProcesso = "";
  String anoProcesso = "";
  String idOrgao = "";
  String codigoRelator = "";
  String interessado = "";
  String assunto = "";

  var state = ProcessosFormPageState.idle;

  var orgaos = <Orgao>[];
  var relatores = <Relator>[];

  bool get isValidRequest => isValid();

  Future _loadResources() async {
    state = ProcessosFormPageState.loading;
    notifyListeners();

    orgaos = await _orgaosRepository.getAllOrgaos();
    relatores = await _relatoresRepository.getAllRelatores();
    state = ProcessosFormPageState.success;
    notifyListeners();
  }

  Future<List<Processo>> searchProcessos() async {…

  bool isValid() {…

  Map<String, dynamic> _setQueryParams() {…
}
```

Figura 7. Classe de Controller da página de busca de processos

A figura acima mostra a implementação do *controller* da tela do formulário de consulta de processos. Nele está implementada a regra de negócio da aplicação, validação dos campos do formulário, preparo de parâmetros a serem enviados na consulta e o envio da consulta, fazendo o intermédio entre a camada de acesso a dados (*ProcessosRepository*) e os componentes de visualização. Um fator relevante a se apontar

dessa implementação é a injeção de dependência utilizando interfaces no construtor. Utilizando uma abstração ao invés de implementação para a comunicação entre a camada de *controller* e a camada de acessos a dados (repositórios), se posteriormente houver necessidade de mudança de busca dos dados, a implementação do repositório ou o pacote utilizado para as requisições, não será necessário mudar nada além da própria implementação do repositório, contanto que a mesma estenda a respectiva interface, sendo assim forçada a seguir o contrato estabelecido para comunicação com a *controller* em questão. Assim respeita-se o princípio da Inversão da Dependência, pois os sistemas mais flexíveis são aqueles em que as dependências de código-fonte se referem apenas a abstrações não itens concretos (MARTIN, 2017). A utilização dessa abordagem torna o código mais testável, flexível e, consequentemente, de mais fácil manutenção. A forma de implementação da injeção de dependência será abordada mais à frente neste trabalho.

Uma pasta que é mostrada na ***Figura 5*** e que não faz parte da abordagem original do MVC é a de *repositories*. Essa pasta diz respeito à implementação do padrão *Repository*, que conceitualmente encapsula os objetos a serem persistidos e as operações a serem realizadas nesses objetos, promovendo uma visão mais orientada a objetos da camada de persistência. O padrão *Repository* também tem como objetivo a separação e dependências de única via entre a camada de domínio e a camada de dados [Fowler 2022]. No escopo inicial de desenvolvimento da aplicação o repositório tem como objetivo a obtenção dos dados junto aos sistemas do TCE. Para esse acesso foi utilizado o package Dio que, segundo sua documentação, "é um poderoso *client* HTTP para o Dart, que suporta *interceptors*, configurações globais, *formdata*, cancelamento de requisições, *download* de arquivos, *timeout* e etc".

```dart
class NoticiasRepository extends INoticiasRepository {
  final Dio dio = Dio();
  final String url = dotenv.env["NOTICIAS_API_URL"]!;

  @override
  Future<List<Noticia>> obterTodasAsNoticias() async {
    var response = await dio.get("$url/latest");

    var todasNoticias =
        (response.data as List).map((n) => Noticia.fromJson(n)).toList();

    return todasNoticias;
  }

  @override
  Future<List<Noticia>> obterUltimasNoticias() async {
    var response = await dio.get("$url/latest");

    var ultimasNoticias =
        (response.data as List).map((n) => Noticia.fromJson(n)).toList();

    return ultimasNoticias;
  }
}
```

Figura 8. Classe da implementação do repositório de notícias

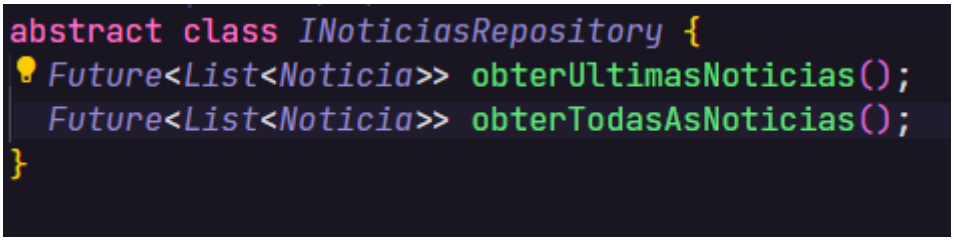

Figura 9. Interface do repositório de notícias

Na ***Figura 8*** pode-se ver o exemplo da implementação de um repositório da aplicação, nesse caso o de notícias. Nele é instanciado o Dio, responsável por fazer as chamadas HTTP para obtenção dos dados a serem utilizados na aplicação. A responsabilidade dele se resume a buscar os dados necessários, transformar esses dados na entidade ao qual aquele repositório se refere e retornar para, no escopo do trabalho, o *controller* e ser transportado para a camada de visualização.

3.1.3 Injeção de Dependência e Gerência de Estado com Provider

A utilização do package Provider é a abordagem de gerenciamento de estado recomendada pela equipe do Flutter. Esse pacote é primariamente um mecanismo de injeção de dependência. Para a notificação da aplicação sobre alguma mudança no estado, deve-se utilizar de outra classe presente no Flutter – *ChangeNotifier* [Slepnev 2020].

Figura 10. Classe de configuração da aplicação com o cadastro dos Providers

A ***Figura 10***, mostra como configuramos as classes a serem providas no sistema. Utilizando o *MultiProvider*, pôde se configurar uma lista de *providers*, disponibilizando essas classes por toda a aplicação com o escopo de *singleton*. Como nossos controllers são os responsáveis por guardar o estado das páginas aos quais eles são responsáveis, portanto, herdando de *ChangeNotifier*, declaram-se com o *ChangeNotifierProvider* o qual identifica para a aplicação que é esse componente que notificará aos *Listeners* (componentes que estão "escutando" as mudanças no estado) quando houver alguma mudança no mesmo.

```dart
class ProcessosForm extends StatefulWidget {
  const ProcessosForm({Key? key}) : super(key: key);

  @override
  State<ProcessosForm> createState() => _ProcessosFormState();
}

You, semana passada | 1 author (You)
class _ProcessosFormState extends State<ProcessosForm> {
  final _processosFormKey = GlobalKey<FormState>();

  @override
  void initState() {
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    var controller = context.watch<ProcessosController>();
    return Scaffold(
      appBar: AppBar(
        title: const Text("Buscar Processos"),
        centerTitle: true,
      ), // AppBar
      body: controller.state == ProcessosFormPageState.loading
          ? const Center(
              child: CircularProgressIndicator(
                color: Color.fromRGBO(45, 179, 112, 1),
              ), // CircularProgressIndicator
            ) // Center
          : SingleChildScrollView(
              child: Container(
                padding: const EdgeInsets.all(10),
                child: Form(
                  key: _processosFormKey,
                  child: Card(
                    elevation: 5,
                    child: Padding(
                      padding: const EdgeInsets.all(20),
```

Figura 11. Widget representando a página do formulário de busca de processos

Na ***Figura 11*** pode-se observar como obtemos a instância cadastrada do *controller* responsável pelo estado da página de consulta de processos. Através do método *context.watch()* obtemos essa instância e como o método é chamado dentro do método

build da nossa página, sempre que houver alguma mudança no estado do *controller*, o método *build* será chamado e a tela será redesenhada respeitando a lógica de negócio referente ao estado modificado.

3.1.4 Outros pacotes utilizados

Alguns outros pacotes externos foram utilizados para a construção da aplicação. O objetivo no uso dos mesmos será mostrado a seguir.

- **font_awesome_flutter**: Pacote utilizado para se obter uma variedade maior de ícones do que os padrões vindos do *Material*. Utilizado principalmente para os ícones presentes nos cards da tela inicial.

- **flutter_dotenv**: Pacote que possibilita o uso de variáveis de ambiente na aplicação, promovendo uma maior flexibilidade para mudanças de ambiente onde a aplicação irá executar (no caso do TCE/RN seriam os ambientes de feature, integration, qa e prod, por exemplo). Através dele cria-se um arquivo *.env* onde estarão os nomes das variáveis utilizadas no sistema e os valores que se deseja utilizar.

- **flutter_html**: Utilizado para exibir código html na tela da aplicação. Utilizado para exibir o texto do corpo da notícia na página de detalhes.

- **carousel_slides**: Utilizado para a construção do carrosel de últimas notícias exibido na página inicial.

- **url_launcher**: Tem como objetivo acessar links externos à aplicação. Utilizado para redirecionar o usuário para as redes sociais do tribunal através dos ícones no topo da página inicial.

- **intl**: Pacote de internacionalização de aplicação. Foi utilizado para a formatação das datas da aplicação para o formato desejado.

3.2 Integração da área restrita com a aplicação mobile

Entrando no escopo da resolução do problema de obtenção dos dados sobre as notícias, tem-se a seguinte situação: os dados das notícias são armazenados em um banco de dados SQL Server chamado de “processos”. Esse banco é consumido por alguns sistemas, dentre eles o da área restrita, onde devido ao modo que foi desenvolvida não provê uma interface que permita a disponibilização das informações para outros consumidores.

Dada a situação descrita acima foi adotada como solução a construção de uma API (*Application Programming Interface*) responsável por nos prover esses dados. O padrão a ser seguido para a construção dessa API será o padrão REST. REST é um acrônimo para *Representational State Transfer*, o qual é definido como um estilo arquitetural de software para implementar *web services* que focam em recursos de sistemas, incluindo como os estados dos recursos são endereçados e transferidos através de HTTP (RODRIGUEZ, 2008).

Algumas diretrizes a serem seguidas para a implementação desse padrão segundo (RODRIGUEZ, 2008) são:

- Usar os métodos HTTP explicitamente
- Não guardar estado
- Expor URIs com estrutura baseada em diretório

- Transferir JSON (*Javascript Object Notation*) ou XML

3.2.1 Arquitetura da API

A arquitetura utilizada no desenvolvimento da API de notícias baseia-se na arquitetura padrão adotada pelo TCE/RN. A estrutura dessa arquitetura está mostrada na **Figura *12***.

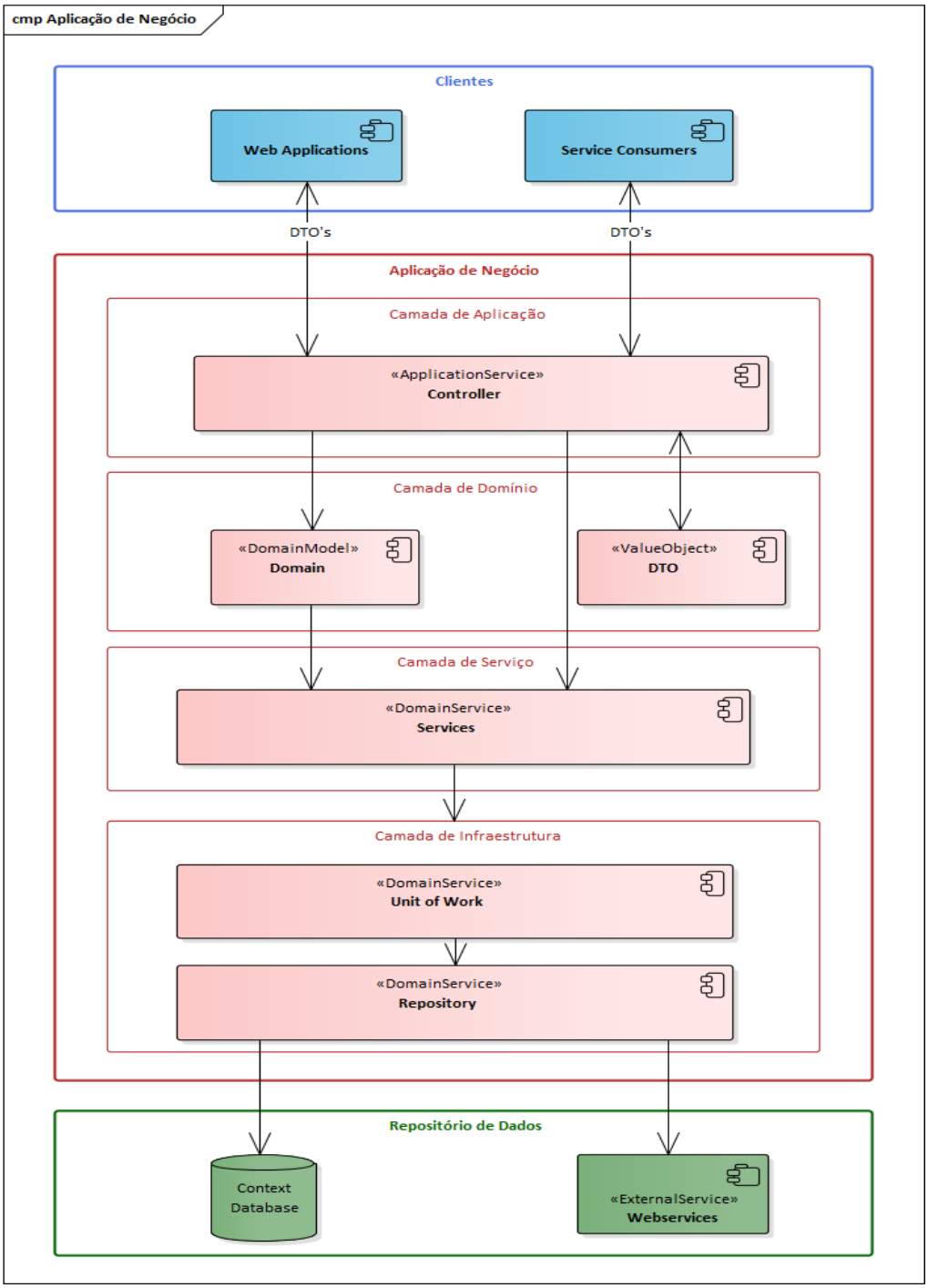

Figura 12. Diagrama de Representação da Arquitetura adotada nas aplicações do TCE/RN (MAIA, 2019)

Foi obtido um template de projeto diretamente do repositório do tribunal que utiliza a estrutura acima. Segundo [Maia 2019], podemos descrever as camadas da seguinte forma:

- Camada de Aplicação: É responsável pela aplicação principal, corresponde ao chamado *application core*, toda aplicação gira em torno dessa camada.

- Camada de Domínio: Responsável pela implementação de classes-models e classes-dto, ou seja, são as classes que representam as entidades de domínio e os objetos de transferências de dados que devem ser utilizados pelos clientes e passados entre as camadas de domínio.

- Camada de serviço: Promove a comunicação entre as diversas camadas. Ela deve implementar as regras de negócio, validações e se comunicar com a camada de repositório de dados ou serviços externos.

- Camada de Infraestrutura: É o canal que deve ser utilizado para realizar operações diretas de manipulação de dados armazenados em banco de dados. Nessa camada estão presentes, por exemplo, as implementações dos repositórios.

Tendo essa visão arquitetural como base, foram feitas adaptações que atendem a necessidade atual do negócio a ser implementado.

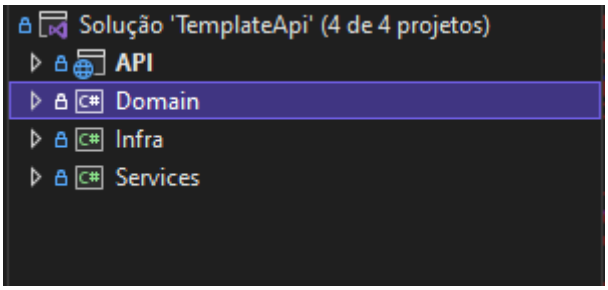

Figura 13. Estrutura de pastas da aplicação aplicando a arquitetura Padrão do TCE/RN

Foram mantidas as camadas originais, porém na implementação a camada de *Service* não foi utilizada devido à simplicidade do escopo atual da aplicação. Apesar de não utilizada, a camada foi deixada no projeto visando uma possível evolução do mesmo, abrangendo mais casos de uso além da consulta na continuidade do projeto.

3.2.2 .NET Core e suas ferramentas

A tecnologia utilizada para o desenvolvimento da aplicação foi o ASP.NET Core, tecnologia pertencente a *stack* principal de desenvolvimento do TCE/RN. Esse framework faz parte da plataforma de desenvolvimento .NET (sendo usada a versão 3.1 nesse projeto), criado pela Microsoft, consistindo em uma estrutura multiplataforma (macOS, Windows e Linux), de alto desempenho e de software livre para a criação de aplicativos modernos, habilitados para nuvem e conectados à Internet. A linguagem utilizada para o desenvolvimento foi o C#, mas a plataforma dá suporte à outras linguagens tais como Visual Basic e F# [ASP.NET docs 2022]

O desenvolvimento da API foi iniciado através da modelagem da entidade de Notícia. Através do acesso ao banco de processo pode-se verificar a existência da tabela Noticias, onde estão armazenados os dados que são consumidos pelo aplicativo móvel. Foi utilizada a instância do banco de processo no ambiente de *hotfix*, o qual é um ambiente clonado do ambiente de produção, de modo a obter dados mais consistentes para o

desenvolvimento da aplicação. Baseado em como está estruturada a tabela Noticias pode-se chegar ao modelo de entidade mostrado na figura abaixo.

```
namespace Domain.Entities
{
    10 referências
    public class Noticia
    {
        [Key]
        2 referências
        public int Id_Noticia { get; private set; }
        1 referência
        public string Titulo { get; private set; }
        3 referências
        public DateTime DataIniVal { get; private set; }
        3 referências
        public DateTime DataFimVal { get; private set; }
        1 referência
        public string ImagemD { get; private set; }
        1 referência
        public string Texto { get; private set; }

        0 referências
        public Noticia() { }

        0 referências
        public Noticia(int idNoticia, string titulo, DateTime dataIniVal, DateTime dataFimVal, string imagemD, string texto)
        {
            Id_Noticia = idNoticia;
            Titulo = titulo;
            DataIniVal = dataIniVal;
            DataFimVal = dataFimVal;
            ImagemD = imagemD;
            Texto = texto;
        }
    }
}
```

Figura 14. Classe que representa a entidade Notícia

Os dados modelados para a entidade Noticia foram baseados nas informações necessárias a serem mostradas na interface no aplicativo mobile desenvolvido.

Após construída a entidade foi definido quais as rotas que seriam necessárias nesse momento para os casos de uso presentes no sistema. Chegou-se às seguintes rotas mostradas na ***Figura 15***:

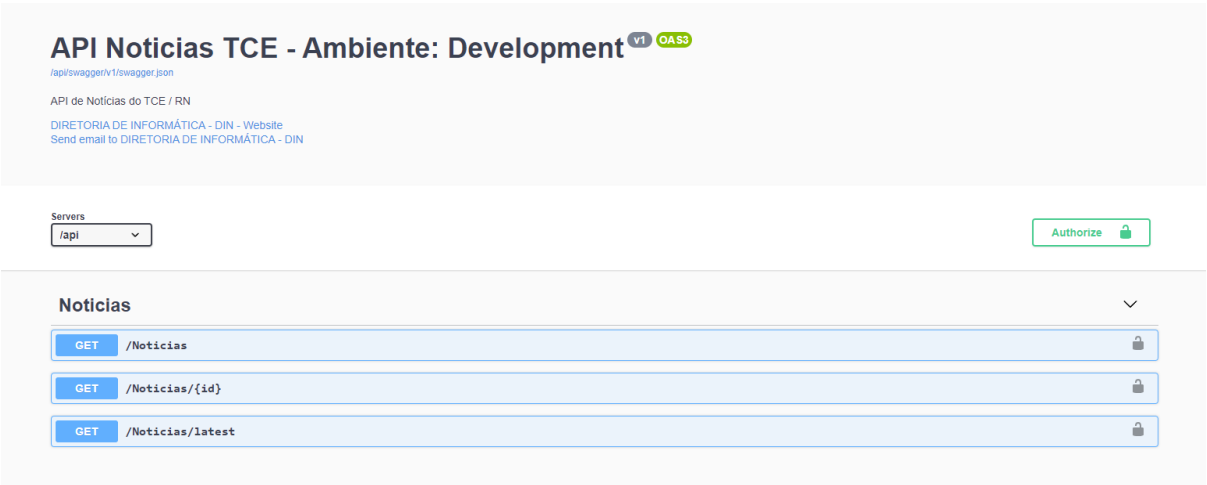

Figura 15. Documentação utilizando Swagger da API de Noticias

- /Noticias: Rota responsável por obter todas as notícias ainda válidas.
- /Noticias/{id}: Rota responsável pela busca de uma notícia recebendo o seu identificador único (id) como parâmetro da rota

- /Noticias/latest: Rota responsável por retornar as 5 notícias mais recentes. Essa rota é utilizada na página inicial da aplicação mobile.

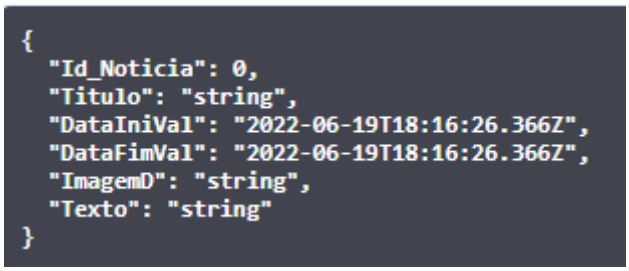

Figura 16. Modelo de resposta da rota /Noticias/{id}

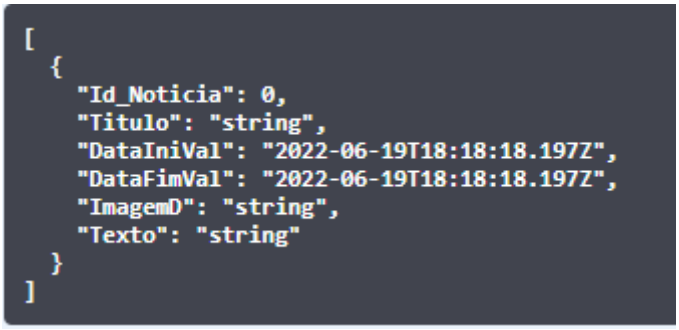

Figura 17. Modelo de resposta da rota /Noticias

Na construção dos *controllers*, os repositórios foram injetados diretamente na classe sem necessidade de passar por uma camada de *service*. Como dito anteriormente, essa adaptação pode ser feita dada a simplicidade atual dos casos de uso da aplicação, não existindo complexidade suficiente que justifique essa abstração.

```
namespace Application.Controllers
{

    [ApiController]
    [Route("[controller]")]
    1 referência
    public class NoticiasController : ControllerBase
    {
        private readonly INoticiaRepository _noticiaRepository;

        0 referências
        public NoticiasController(INoticiaRepository noticiaRepository)
        {
            _noticiaRepository = noticiaRepository;
        }


        /// <response code="200">Retorna todas as notícias</response>
        /// <response code="401">not authorized</response>
        /// <response code="500">internal error server</response>
        [ProducesResponseType(typeof(List<Noticia>), 200)]
        [HttpGet]
        0 referências
        public async Task<IActionResult> GetAll()
        {
            return Ok(await _noticiaRepository.ObterTodas());
        }

        [ProducesResponseType(typeof(Noticia), 200)]
        [HttpGet("{id:int}")]
        0 referências
        public async Task<IActionResult> ObterDetalhesNoticia(int id)
        {
            var noticiaEncontrada = await _noticiaRepository.ObterPorId(id);

            if (noticiaEncontrada is null)
            {
                return NotFound();
            }

            return Ok(noticiaEncontrada);
        }

        [ProducesResponseType(typeof(List<Noticia>), 200)]
        [HttpGet("latest")]
        0 referências
        public async Task<IActionResult> ObterUltimasNoticias()
        {
            var noticiaEncontrada = await _noticiaRepository.ObterUltimasNoticias();

            return Ok(noticiaEncontrada);
        }

    }
}
```

Figura 18. Classe Controller para os casos de uso de notícias

Na implementação mostrada acima é mantida a abordagem de se respeitar o Princípio da Inversão de Dependência, assim como na aplicação mobile. Para a implementação dos repositórios, foi utilizada o pacote *Entity Framework* como ORM

(*Object Relational Mapper*), responsável por fazer a ponte entre os dados do banco de dados e as entidades da aplicação. A configuração dessa ferramenta já é configurada por padrão no template padrão do TCE, necessitando apenas ser configurado o contexto do banco de dados e sua string de conexão. Respeitando a divisão de camadas da arquitetura padrão do tribunal essa implementação é feita na camada de Infra, deixando o domínio da aplicação agnóstico de pacotes externos.

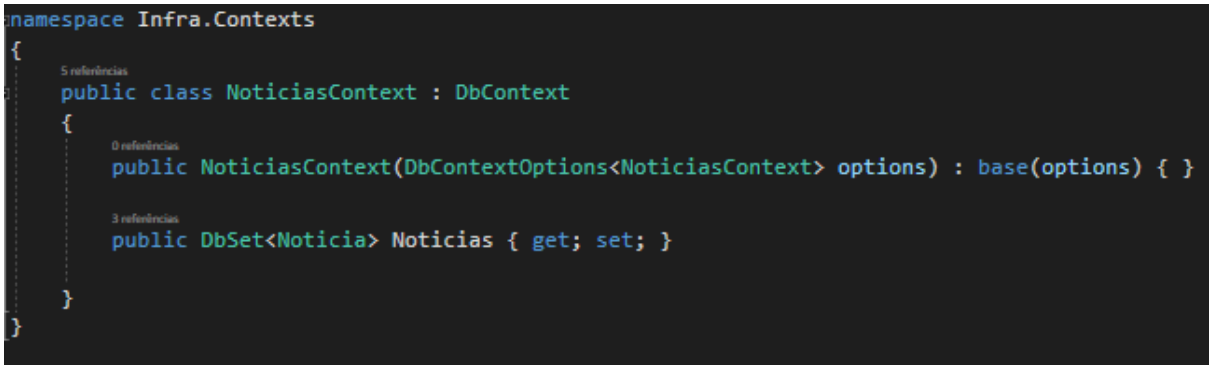

Figura 19. Classe de implementação do contexto do banco de dados da aplicação utilizando Entity Framework

4. Dificuldades Encontradas

Durante o desenvolvimento da solução abordada neste trabalho é importante destacar algumas dificuldades que surgiram durante o processo e que justificam algumas abordagens utilizadas e podem impactar os resultados apresentados.

A primeira das dificuldades foi com relação à fidelidade da construção do layout e seus componentes. Devido a falta da existência de um *mock up* ou protótipo de telas para servir de base para o desenvolvimento, a técnica utilizada foi puramente percepção visual do desenvolvedor. Era aberto o aplicativo do TCE RN Digital que está atualmente disponível para uso público e, de acordo com o que era visualizado, era construída a tela. Essa abordagem afeta de maneira considerável o resultado apresentado se considerarmos como critério a comparação de aspectos como cores, fontes, ícones e etc. Procurou-se ferramentas que fizessem o layout ficar o mais próximo possível do objetivo.

Outra dificuldade encontrada foi com relação ao funcionamento do emulador mobile Android utilizado (AVD Manager do Android Studio) para o desenvolvimento com as *urls* dos sistemas internos do TCE. Isso se deve a necessidade da utilização de uma VPN (*Virtual Private Network*) para o acesso às mesmas, pois esses sistemas internos utilizados permitem acesso somente com a rede privada do tribunal a fim de proteger a propriedade intelectual presente nos mesmos. Tendo isso em vista o emulador não conseguia acessar as urls dos sistemas de Processos e do TCE Admin. Para o desenvolvimento do módulo de processos da aplicação mobile, que faz a utilização desses serviços internos, foi compilada a aplicação Flutter para a plataforma Windows a fim de validar o funcionamento das chamadas HTTP feitas para o funcionamento correto da aplicação. Para o desenho do layout foram utilizados dados aleatórios e fixos, visando o máximo de fidelidade de visual para a interface mobile, tendo em vista que o desenvolvimento do layout compilando para Windows, mesmo que adaptando o tamanho da tela, poderia gerar diferenças devido à particularidades da plataforma.

E por fim, a instabilidade em alguns momentos da aplicação atualmente em produção e modificações feitas posteriormente ao início do desenvolvimento deste trabalho foram desconsideradas.

5. Resultados

Como resultado do trabalho descrito neste artigo temos a construção da API para a realização da integração das informações de notícias como mostrado anteriormente neste trabalho. Essa implementação contribui para o crescimento dos serviços oferecidos pelo tribunal, facilitando uma possível integração dessas informações em outros serviços existentes e que serão desenvolvidos. A forma que foi construído, obedecendo o padrão já estabelecido na instituição diminui a curva de aprendizado no que diz respeito ao que foi desenvolvido visando a continuidade das implementações para o sistema por parte da equipe de desenvolvimento do tribunal.

O outro resultado a ser demonstrado refere-se à construção da interface mobile para o TCE RN Digital. A seguir serão mostradas algumas das principais telas construídas visando o objetivo do trabalho.

Figura 20. Tela inicial da aplicação em Flutter

A ***Figura 20*** mostra a tela inicial da aplicação móvel desenvolvida. Como dito anteriormente, foi tido como base o layout já existente na aplicação mantida pela empresa terceira.

Figura 21. Tela de listagem de notícias da aplicação Flutter

A ***Figura 21*** consiste na tela de listagem de todas as notícias ativas no TCE. Essa tela pode ser considerada crítica de modo que a quantidade de dados que são recebidos para serem mostrados é grande. Esse fator poderia ter impacto considerável na performance da aplicação, mas o *Flutter* nos traz recursos que minimizam esse impacto. Na tela apresentada foi utilizado o *widget ListView.builder()*, o qual tem como objetivo carregar as informações mostradas na tela sob demanda, ou seja, os demais dados que são mostrados na rolagem da tela pelo usuário só são carregados ao entrarem na visualização do aplicativo.

Figura 22. Tela do formulário de busca de processos

A ***Figura 22*** mostra a tela de formulário a ser mostrada quando o usuário acessa o módulo de consulta processual. Nessa tela são utilizados os sistemas do TCE Admin e a API de Processos do tribunal a fim de buscar os dados dos campos Órgão e Relator que podem ser utilizados como filtro da busca. A validação do formulário é feita de maneira reativa utilizando o widget *ValueNotifier*, o qual herda de *ChangeNotifier* nos permitindo controlar, por exemplo, se o botão de consultar fica habilitado ou não ao usuário a partir dos valores dos campos do formulário a cada mudança no estado do mesmo.

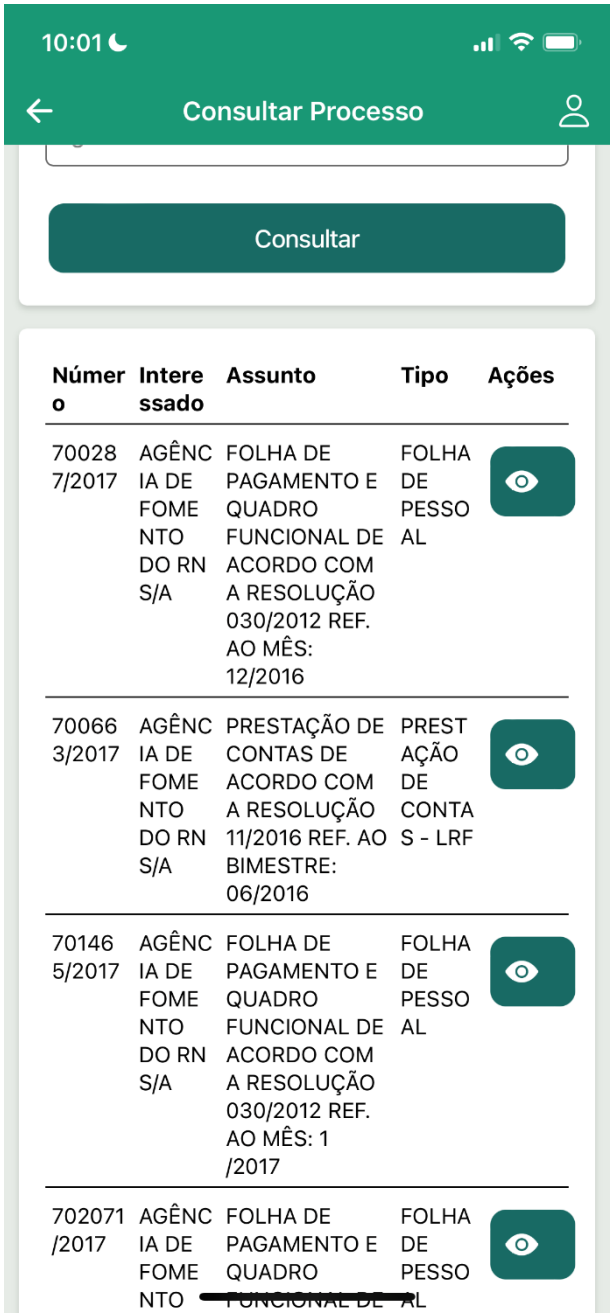

Figura 23. Tela de listagem de resultados da aplicação atual do TCE RN Digital

Figura 24. Tela de listagem dos resultados da busca de processos no aplicativo Flutter

Diferente de como é feito na aplicação atual (***Figura 23***), a área de visualização dos resultados da busca de processos foi colocada em uma página a parte (***Figura 24***) de modo a melhorar a visualização das informações que estavam posicionadas de um modo não muito agradável visualmente, prejudicando a experiência do usuário. O atributo "Tipo" que era mostrado na tela originalmente foi retirado a fim de melhorar a visualização da lista para o usuário.

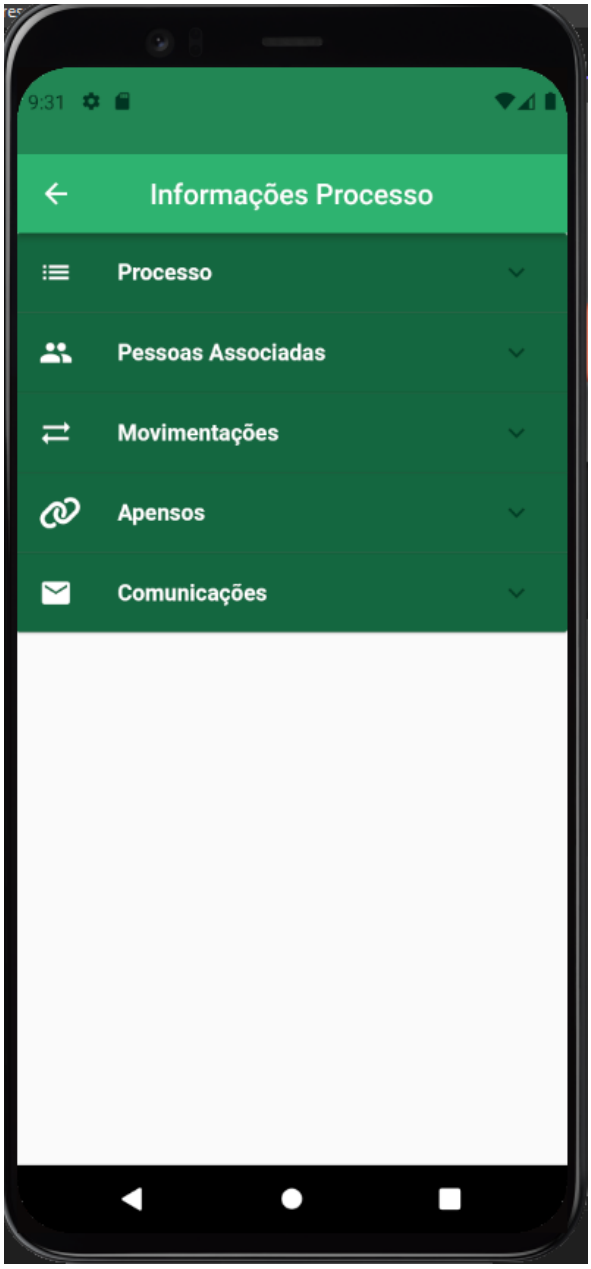

Figura 25. Tela de detalhes do processo

Figura 26. Tela de detalhes do processo, destacando a aba de processo

A ***Figura 25*** e a ***Figura 26*** mostram as telas que não existiam anteriormente na aplicação, a área de visualização dos detalhes do processo, que existiam somente na interface web do tribunal. Para a construção do layout da aplicação móvel procurou-se manter a identidade visual da página web utilizada pela aplicação em produção atualmente, necessitando-se ainda da definição do formato de visualização de abas contendo dados mais extensos e complexos como a de Comunicação.

6. Considerações Finais

Este artigo procurou mostrar o processo de desenvolvimento do aplicativo para o tribunal, mostrando o processo em todas as suas etapas: entendimento da demanda, levantamento de requisitos, definição de arquitetura, ferramentas a serem utilizadas, dificuldades encontradas e os resultados no escopo de interface visual, desenvolvimento e uso de sistemas internos. Algumas decisões de layout foram tomadas, principalmente em relação

a telas que não existiam na aplicação. Procurou-se utilizar das melhores práticas de desenvolvimento tanto para a interface mobile como para a API de integração visando uma melhor manutenção e legibilidade do código, tendo em vista inclusive a integração dos módulos desenvolvidos com os demais módulos da aplicação. Os padrões aplicados foram escolhidos pensando nas abordagens já utilizadas no dia a dia de desenvolvimento do tribunal, buscando suavizar a entrada de pessoas que não tenham tanta experiência com o ambiente mobile. A falta de um padrão de desenvolvimento para aplicações desse tipo foi um fator com impacto grande principalmente no início do projeto, mas que gerou reflexões positivas sempre pensando em aproximar o que foi desenvolvido ao que já é feito no dia a dia do tribunal.

Visando os próximos passos no desenvolvimento do projeto têm-se os pontos prioritários:

- Integração dos módulos desenvolvidos com os demais módulos (Painel de Obras, Escola de Contas, Painel da Transparência, Pauta das Sessões e Ouvidoria), que necessitará de um estudo para entender quais padrões serão mantidos e quais serão modificados no escopo de uma aplicação maior.

- Definição e padronização do layout, pois a falta de um *mock up* como guia do desenvolvimento da interface gera algumas inconsistências de layout e até duplicação de código dentro da aplicação.

- Definição da visualização das entidades referentes aos detalhes do projeto.

A utilização do *Flutter* para o desenvolvimento, de início, foi uma quebra de paradigma considerável devido à abordagem declarativa de desenvolvimento, mas que com o passar do tempo se mostrou uma forma bastante produtiva de desenvolver que gera uma excelente experiência de desenvolvimento. A documentação é muito rica e bem escrita, tanto do framework quanto dos pacotes externos, facilitando seu uso.

Com relação ao desenvolvimento da API de integração, a existência de um padrão de arquitetura foi um grande facilitador para o desenvolvimento da aplicação. A simplificação das abstrações utilizadas, como a não utilização da camada de *service*, foi feita pensando na complexidade atual do projeto mas isso não quer dizer que haja rigidez para a mudança num possível crescimento deste serviço.

Por fim, o maior impacto com a realização desse trabalho está em trazer inovação para o ambiente de desenvolvimento do tribunal com o desenvolvimento focado em uma nova plataforma, diminuindo os custos do tribunal para a manutenção dos seus serviços e, ao trazer o desenvolvimento do projeto para dentro do TCE/RN, tornamos mais ágil e flexível a entrega de seus serviços na plataforma mobile sempre visando a necessidade da sociedade em geral.

7. Referências

Mark, R. e Neal, F. (2020), Fundamentals of Software Architecture, O`Reilly Media, Inc., 1º Ed.

Neves, J., & Junior, V. M. (2020). “Uma análise comparativa entre flutter e react native como frameworks para desenvolvimento híbrido de aplicativos mobile: Estudo de caso visando produtividade”. Ciência da Computação-Tubarão.

da Silva, M. M., & Santos, M. T. P. (2014). Os paradigmas de desenvolvimento de aplicativos para aparelhos celulares. Revista TIS, 3(2).

Flutter (2022) [Internet]. Flutter architectural overview. [Acessado em 18 de junho de 2022]. Disponível em https://docs.flutter.dev/resources/architectural-overview

Faust, S. (2020). Using Google s Flutter Framework for the Development of a Large-Scale Reference Application (Doctoral dissertation, Hochschulbibliothek der Technischen Hochschule Köln).

Daoudi, A., ElBoussaidi, G., Moha, N., & Kpodjedo, S. (2019, April). An exploratory study of mvc-based architectural patterns in android apps. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (pp. 1711-1720).

Repository (2022) [Internet]. Repository. [Acessado em 18 de junho de 2022]. Disponível em https://martinfowler.com/eaaCatalog/repository.html

Slepnev, D. (2020). State management approaches in Flutter.

MAIA, G. L (2019). Uma proposta de Arquitetura de Referência para Sistemas de Informação do TCE-RN. Trabalho de Conclusão de Curso (Especialização em Residência Tecnologia da Informação aplicada à Área Jurídica) - Centro de Ciências Exatas e da Terra, Universidade Federal do Rio Grande do Norte, Natal.

ASP.NET Core (2022) [Internet]. Visão geral do ASP.NET Core. [Acessado em 18 de junho de 2022]. Disponível em https://docs.microsoft.com/pt-br/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

RODRIGUEZ, Alex. Restful web services: The basics. IBM developerWorks, v. 33, p. 18, 2008.

MARTIN, R. (2017), Clean Architecture: A Craftsman`s Guide to Software Structure and Design, Pearson Education, Inc., 1º Ed.

Palmieri, M., Singh, I., & Cicchetti, A. (2012, October). Comparison of cross-platform mobile development tools. In 2012 16th International Conference on Intelligence in Next Generation Networks (pp. 179-186). IEEE.