**UNIVERSIDADE FEDERAL DE MINAS GERAIS**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

Helio Victor Flexa dos Santos

# DCC App: Um Aplicativo do DCC Construído Usando Boas Práticas e Técnicas de Engenharia de Software

Belo Horizonte
2023

Helio Victor Flexa dos Santos

# DCC App: Um Aplicativo do DCC Construído Usando Boas Práticas e Técnicas de Engenharia de Software

**Versão Final**

Projeto apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de bacharelado em Ciência da Computação.

Orientador: Marco Túlio de Oliveira Valente
Coorientadora: Luciana Guimarães Carvalho

Belo Horizonte
2023

# Resumo

Nessa documentação, detalharemos o processo de estudo, aprendizado e implementação de um software de produção desenvolvido com base nas boas práticas e técnicas de Engenharia de Software. Nesse contexto, explicitamos todo mapeamento feito na segunda etapa do projeto, com a criação da tela de oportunidades e a suíte de testes.

**Palavras-chave:** engenharia de software, boas práticas, engenharia de software moderna, computação.

# Lista de Figuras

# Sumário

# Capítulo 1

# Introdução

Nos tempos atuais, os softwares estão presentes no dia-a-dia das pessoas, seja eles para fins de trabalho, estudo ou lazer. Com esse uso intenso no mundo atual, a qualidade das aplicações é um fator determinante para que ele tenha destaque nos demais no ambiente de desenvolvimento de software. Um bom planejamento do projeto, boas práticas de programação, o uso de frameworks estáveis, bons testes, uma boa arquitetura e outros, podem ser considerados fatores primordiais para que a excelência do software seja alcançada e com ela a popularidade por parte dos consumidores finais seja alcançada gerando retorno positivo para a empresa.

Para que essa qualidade esteja presente na sua aplicação é necessário que desde a concepção de esboços, instalação de todo ambiente, implementação, manutenção e evolução, seja elaborado seguindo as métricas e qualidades debatidas da área de Engenharia de Software, entretanto, em um cenário real por diversos fatores desde a inexperiência da equipe, contratempos e adversidades, muitas dessas práticas são deixadas de lado ou até mesmo abandonadas, ocasionando em problemas ao longo do projeto como bugs, alterações inesperadas de compatibilidade ou mudanças abruptas nos requisitos do sistema.

Sendo assim, neste projeto desenvolvemos um aplicativo, DCC App, ao qual pretende seguir a construção concreta de um aplicativo na visão do que a área prega em termos acadêmicos, sendo desde a idealização do produto, planejamento das funcionalidades, desenvolvimento da arquitetura, padrões de projeto, testabilidade, manutenibilidade e evolução do software.

É extremamente comum que empresas de desenvolvimento de software não sigam ou contemplem alguns conceitos empregados pela Engenharia de Software, seja por limitação de escopo ou tempo do projeto. Com isso, é possível perceber que durante a fase de desenvolvimento essas equipes acabam tendo alguns problemas recorrentes já descritos anteriormente como mudanças abruptas nos requisitos por falta de alinhamento com os clientes, problemas de definição de projeto que impactam na construção da arquitetura ou na extensão de alguma funcionalidade.

Em relação a isso, é importante entender o custo de acelerar o desenvolvimento e não realizar alguma das etapas consideradas chaves na Engenharia de Software como: requisitos do sistema, modelagem, princípios e padrões de projeto, arquitetura, testes

podem ter o custo alto a longo prazo para a equipe, seja com atrasos, implementações que podem ser refeitas, alterações de escopo e etc. Sendo assim, nosso objetivo aqui é mostrar que percorrer o passo a passo das técnicas e boas práticas que a área propõe irá atenuar problemas que ocasionalmente poderiam ocorrer caso o cenário real comum fosse usado.

Dado o exposto, nessa primeira parte do projeto iremos descrever os resultados alcançados até esse ponto de corte, descrevendo todo percurso realizado e eventuais problemas que foram amenizados devido ao prévio planejamento. Essa documentação está organizada da seguinte forma: na seção 2 iremos expor os trabalhos relacionados ao desenvolvido nesta documentação. Já na seção 3 iremos relembrar as etapas produzidas na fase 1. Na seção 4 iremos relembrar os trabalhos que foram definidos na fase anterior e que foram executados nesta etapa do projeto. Na seção 5, iremos detalhar todo o processo de desenvolvimento do aplicativo DCC App nesta segunda parte. Na seção 6 iremos apontar as adversidades encontradas e quais tarefas que estavam planejadas e não foram feitas. Por fim, na seção 7 iremos concluir o projeto realizado nessa primeira parte.

# Capítulo 2

# Trabalhos Relacionados

Nessa seção abordaremos todo o conteúdo didático utilizado como orientação para a elaboração desse projeto, seguindo o proposto.

Desenvolver um projeto prático requer um certo cuidado, pois nele não estamos expondo algum tipo de investigação ou alguma pesquisa mas sim aplicando algum tipo de conceito presente na área. Sendo assim, visamos aplicar a base teórica da Engenharia de Software sobre um produto final, traçando um roteiro do passo a passo para a elaboração de um software seguindo as boas práticas e técnicas da área.

Como citado anteriormente, a popularização dos meios tecnológicos faz crescer o desenvolvimento de novas aplicações de diversos segmentos no mercado, porém, esse crescimento impacta na qualidade dos produtos disponibilizados. É possível notar que em empresas de médio e pequeno porte, a preocupação maior é com o produto final em seu estado puro e funcional, mas note que funcional não significa que esse desenvolvimento está atrelado com as definições da Engenharia de Software.

Para esse projeto, nossa direção foi de garantir que todo o desenvolvimento esteja atrelado com as boas práticas e técnicas, seja pela arquitetura, padrões de projeto, testabilidade, manutenção entre outros conceitos empregados pela área. Para isso, uma das ideias seguidas nesse projeto foi debatido por [4]. Os autores enfatizam a falta de prática acadêmica para lidar com problemas relacionados a Engenharia de Software, isto é, durante o curso não há possibilidade dos alunos vivenciarem problemas desafiadores e que de fato são rotineiros na indústria, pois a carga prática final é controlada apenas para exercitar o conteúdo visto.

Sendo assim, em paralelo a proposta apresentada por Balaban da criação de um **SE-lab**, procuramos nos guiar pelo conceito por trás do laboratório para desenvolver o projeto onde nesse caso, aplicando as técnicas e boas práticas não teríamos a disposição um ambiente controlado e sem grandes problemas já que estamos desenvolvendo algo para produção, que será descrito na seção 3. Assim aplicamos todos os conceitos reforçando a carga teórica, entretanto, mantendo um ambiente similar ao real com desafios ao longo do desenvolvimento.

Também, para nos guiar em relação a todo conteúdo teórico da Engenharia de Software, utilizamos o livro **Engenharia de Software Moderna** do autor [6]. Nesse

livro, temos um passo a passo para a elaboração de um software seguindo os princípios da área. Baseado integralmente nesse livro, desenvolvemos a nossa aplicação a partir de passo descrito na ordem em que os capítulos foram expostos, vide 3. Com isso, todas as fases do desenvolvimento percorreram os conceitos exercitados no livro ao qual é amplamente utilizado pela comunidade acadêmica e uma referência no que tange aos desafios e práticas da Engenharia de Software no seu caráter mais moderno utilizado pela indústria atualmente.

Neste guia elaborado por [6], encontramos o chamado **Padrões de Projeto**, no qual são conceitos que aplicamos ao projeto do software trazendo alguns benefícios, como economia de tempo e facilidade no entendimento do código. Logo, para o nosso projeto utilizamos o padrão chamado **Mediador** que também é descrito no livro de Padrões de Projeto, [2], e que será detalhado na seção 3.

Também, está incluído no desenvolvimento a garantia de que o software funcione, isso se dá por meio de testes. Para essa versão inicial do aplicativo, como grande parte da estrutura é composta por mais iterações com o RSS do que lógica de negócio. Vladimir Khorikov, [3], em seu livro diz que a estrutura dos testes em projetos com esse aspecto de "CRUD"tem um formato losangular ao invés da conhecida pirâmide de testes na qual os testes de unidade tem uma maior porcentagem em relação aos de integração e sistema. Para Vladimir, nesse tipo de projeto, testes de integração têm a maior porcentagem comparado aos outros, já que a interação com o banco de dados/API é muito maior que a própria lógica de negócio do código, tendo então esse aspecto losangular.

Com isso, evidenciamos diversas fontes da literatura que foram utilizados para a base do desenvolvimento desse projeto bem como quais são os pontos de interesse a serem explorados para evidenciar como os conceitos da literatura e a aplicação prática podem caminhar em conjunto, de modo que seja possível seguir o que direciona a Engenharia de Software sem que haja algum tipo de perda conceitual ao longo do desenvolvimento.

# Capítulo 3

# A fase 1 do projeto

Nessa seção iremos relembrar todo trabalho já desenvolvido na fase 1 do projeto do aplicativo a fim de situar o leitor sobre as novas concepções.

## 3.1 Requisitos do Sistema

Requisitos "definem o que um sistema deve fazer e quais restrições. Requisitos relacionados com a primeira parte dessa definição — o que um sistema deve fazer, ou seja, suas funcionalidades — são chamados de **Requisitos Funcionais**. Já os requisitos relacionados com a segunda parte — sob que restrições — são chamados de **Requisitos Não-Funcionais**."[6] Para que isso seja definido, temos a ideia do "cliente", uma pessoa, organização ou empresa que irá utilizar o sistema. Essa entidade, entre todos que participam do projeto, é quem melhor consegue responder a pergunta no início desta seção.

Assim, na fase anterior seguindo as boas práticas da Engenharia de Software, após diversas reuniões com o cliente, modelamos as funcionalidades que o software deveria ter, chegando à seguinte concepção do que seria o produto.

*"O aplicativo DCC News a ser desenvolvido tem como cerne da sua concepção um espaço onde os alunos consigam acompanhar as principais informações do Departamento de Ciência da Computação ao longo da sua graduação, isto é, acompanhar as notícias e atualizações do departamento, acompanhar o andamento de eventos interno ou de parceria com a unidade, se situar a cerca de dúvidas previamente já computadas, uma espécie de FAQs, saber quais oportunidades de iniciação científica com bolsas e voluntárias tem disponível e etc."*

Após essa ideia inicial, foi possível definir através das histórias de usuário, técnica do modelo de desenvolvimento ágil **Scrum** [5], as tarefas principais que se havia concepção naquela altura sobre o projeto.

Por fim, a última métrica do roteiro de requisitos do sistema disponível no livro


Figura 3.1: Histórias de usuário

"Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade."[6] são as tecnologias utilizadas.

Para o DCC News, como a aplicação é voltada para o contexto mobile, estamos utilizando nessa primeira fase, como framework **react-native** juntamente com o banco de dados relacional offline **realm**.

## 3.2 Modelagem

Definido os requisitos do sistema, seguimos para o próximo passo do roteiro prático de boas práticas e técnicas de Engenharia de Software. Nessa fase utilizamos a técnica de **Diagrama de Classes** para representar uma possível estrutura do nosso sistema.

Durante o processo de modelagem e já visando as implementações futuras, separamos a estrutura do projeto em suas partes, API e Servidor, cabe aqui relembrar a diferença entre eles. Assim, utilizamos a camada de API para fazer o consumo dos dados

de notícias, funcionalidade principal dessa fase 1.

- **API**: consumimos os dados, como por exemplo o feed de notícias, através de um ponto de acesso em XML, provenientes do RSS. Nesse caso os dados são dispostos somente dessa fonte, não precisando ser estruturados.
- **Servidor**: outro ponto de consumo de dados será por meio do servidor, nesse caso, alguns dados serão armazenados no banco de dados local ou remoto e utilizaremos a comunicação com esse para requisitar os conteúdos necessários, assim podemos estruturá-los e apresentá-los aos usuários no sistema.

Outro ponto da modelagem foi a definição do padrão de projeto [2] utilizado no sistema. Após estudos e como já planejado na modelagem a adição de novas funcionalidades não contempladas na primeira fase do projeto utilizando o servidor, decidimos utilizar o padrão conhecido como **Mediador**, assim, toda essa interface é abstraída, já que os métodos irão realizar apenas a chamada para a classe com o tipo de dado solicitado, ficando a cargo dele verificar qual o local que a requisição deve ser feita, sem que haja preocupação por parte dos métodos mais externos com esse tipo de conhecimento.

Podemos ver abaixo, a modelagem do DCC News utilizando o mediador.

## 3.3 Arquitetura

Recapitulando, já temos conhecimento sobre quais funcionalidades o aplicativo terá, também já sabemos como é a estrutura central com suas requisições e consumo de dados. Para completar esse ciclo precisamos definir a arquitetura utilizada pelo nosso projeto.

Arquitetura de software se preocupa com o projeto no mais alto nível. Ela inclui as decisões de projeto mais importantes em um sistema e que dificilmente serão revertidas futuramente. Assim, mediante a nossa modelagem, o projeto utiliza a arquitetura do tipo **MVC**.

Com a adoção desse padrão arquitetural, trazemos consigo alguns benefícios: podemos utilizar o mesmo modelo em diferentes visões, possibilitando criar especializações de trabalho no desenvolvimento, assim, para a manutenção do código podemos ter pessoas capazes de lidar somente com alguma parte do sistema, também, favorece a testabilidade do projeto. [1]

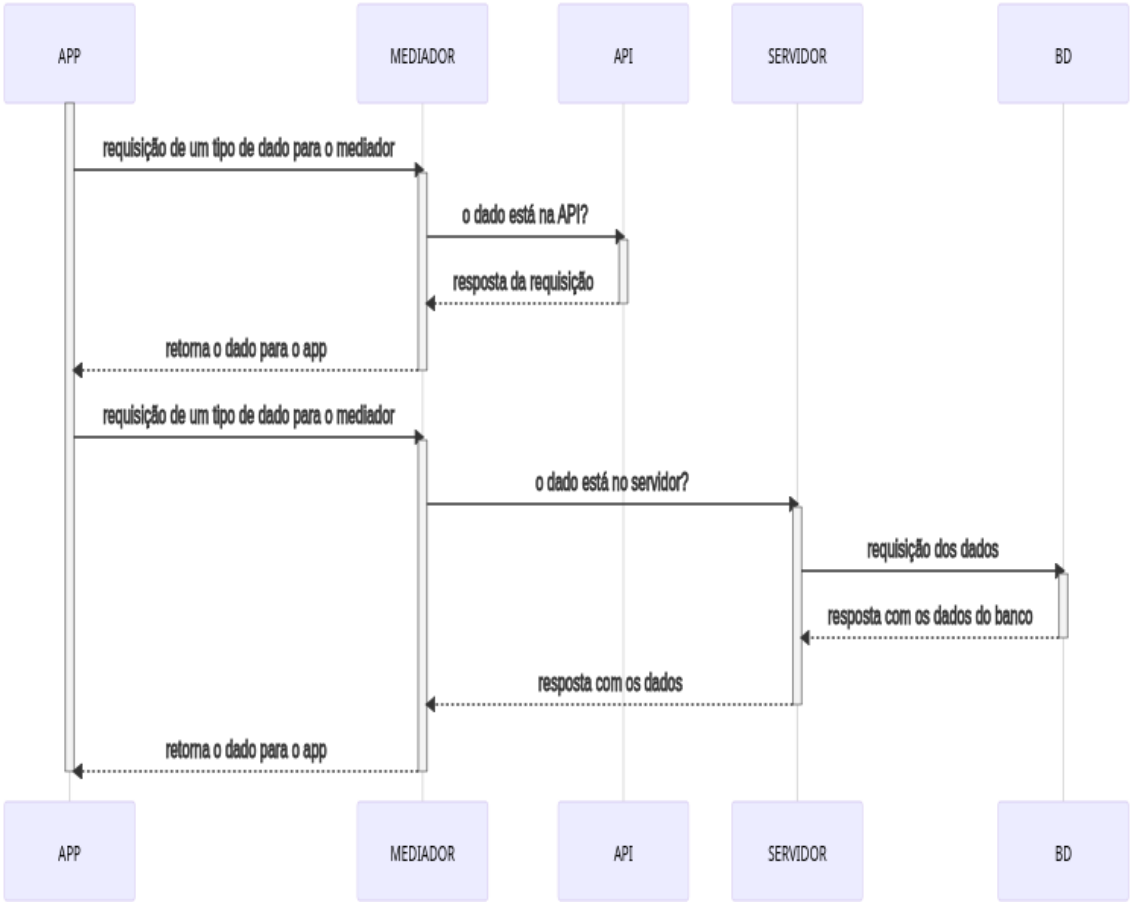



Figura 3.2: Modelagem do DCC News

Com isso, concluímos o roteiro inicial para o desenvolvimento do projeto nos moldes da Engenharia de Software. A partir desse momento e de posse de todo esse arsenal, podemos enfim iniciar o desenvolvimento da aplicação.

## 3.4 Desenvolvimento

Diante de todo conhecimento disponível, a partir deste ponto iniciamos o desenvolvimento da aplicação de fato. Como já citamos anteriormente, o projeto está utilizando o desenvolvimento ágil Scrum [5], logo o uso de sprints se faz necessário, juntamente com as histórias de usuários e casos de uso que já foram citados.

Nessa primeira parte, foram realizadas 3 sprints. Em cada uma delas, como segue as técnicas da Engenharia de Software, foram aplicados os conceitos referentes ao desenvolvimento de métodos ágeis, então, a cada sprint tivemos um número limitado de histórias do usuário para desenvolver, limite esse que não poderia ser maior que a quanti-

dade de story points de cada tarefa, que nesse contexto não foi medido por essa métrica já que havia apenas uma pessoa desenvolvendo, onde no nosso contexto foi delimitado pela dificuldade das tarefas definidas em cada sprint.

### 3.4.1 As dificuldades dos testes da fase 1

Software é uma construção complexa, assim é de entendimento que erros e inconsistências podem acontecer das mais variadas formas. Para isso, uma boa prática da Engenharia de Software é realizar bons testes para atenuar esses problemas.

Vladimir Khorikov, [3], em seu livro diz que a estrutura dos testes em projetos com aspecto de "CRUD"tem um formato losangular ao invés da conhecida pirâmide de testes na qual os testes de unidade tem uma maior porcentagem em relação aos de integração e sistema. Para Vladimir, nesse tipo de projeto, testes de integração têm a maior porcentagem comparado aos outros, já que a interação com o banco de dados/API é muito maior que a própria lógica de negócio do código, tendo então esse aspecto losangular.A partir desse entendimento, iniciamos a modelagem da nossa suite de testes com o foco em testes de integração. Porém, também temos conhecimento que algumas dificuldades existem em testar os componentes criados no react-native. Isso acontece, pois uma simples estrutura utiliza diversos artifícios de implementação para funcionar corretamente, sendo assim, seria necessário utilizar técnicas de **teste double** para que os componentes sejam testados.

Testes de integração no âmbito do projeto, não envolvem chamadas com o banco de dados especificamente, já que as requisições em sua maioria nessa primeira fase são realizadas via endpoints RSS. Assim, seria necessário que além dos testes de integração para verificar essas requisições, também tivéssemos testes de renderização de componentes que para esse framework também podem ser vistos como da classe de integração.

Entretanto, como definido pelas boas práticas da Engenharia de Software, ao desenvolver uma tarefa, seu conjunto de testes também seria elaborado juntamente com elas, porém por conta do que foi descrito anteriormente, houve uma grande dificuldade de que eles sejam escritos de forma correta, já que os próprios componentes do framework possuem certas dificuldades, logo durante essa primeira parte do projeto, em alguns pontos específicos do projeto realizamos os chamados **testes manuais** que também são utilizados na área, porém em um grau menor.

### 3.4.2 Implementação dos testes alfa

Após a descrição anterior sobre o desenvolvimento dos testes, partimos para a próxima etapa: **testes de aceitação**.

Testes de aceitação são aqueles que servem para verificar se o sistema que foi desenvolvido está em concordância com as necessidades dos clientes. Eles possuem duas características: são testes manuais feitos pelos clientes do produto e também envolve um processo de validação do sistema, logo, caso tudo esteja de acordo, o produto entra para produção.

Esses testes podem ser divididos em duas fases: a primeira é chamada de **teste alfa**, esses são realizados com um grupo restrito de usuários em um ambiente controlado e com acompanhamento dos desenvolvedores. Após a aprovação temos a segunda fase chamada de **testes beta**, nessa fase não temos mais um ambiente controlado, o grupo de usuários é maior e não há acompanhamento dos desenvolvedores. Diante disso e caso aprovado o software finalmente entra em produção.

Para o DCC News, também temos os testes de aceitação. Como proferido no cronograma desenvolvimento na fase inicial do projeto, ao final da terceira sprint teríamos a disponibilização de um MVP, mínimo produto viável [6], que consiste em uma versão inicial do aplicativo com funções mínimas para validação dos usuários.

Assim, seguindo as técnicas da Engenharia de Software, estamos atualmente na primeira fase dos testes de aceitação, na qual disponibilizamos para o cliente e um pequeno grupo de usuários selecionados uma versão *preview* do aplicativo para que esses possam validar suas funções, fluxo de execução, interface, além de sugerir mudanças e melhorias na aplicação.

### 3.4.3 Disponibilização na Play Store

Como citado acima, estamos na primeira fase dos testes de aceitação. Assim, para disponibilizar as versões chamadas de *preview* para a comunidade, a loja de aplicativos Play Store foi utilizada nesse primeiro momento. Com isso, todos os usuários que fazem parte do teste alfa podem reportar feedback do aplicativo diretamente pela loja, além das opções presentes na interface do sistema, assim logo que temos conhecimento podemos avaliar e ajustar o que for necessário.

Dado o cenário, é possível ter o ambiente controlado como a própria definição de testes alfa retrata, mantendo um contato direto com os usuários para constantemente

melhorar a experiência de uso da aplicação.

# Capítulo 4

# Trabalhos planejados para a fase 2

Como descrito na seção anterior, podemos relembrar de todo o caminho traçado na fase 1 do projeto até o ponto de partida para essa segunda fase. Situamos nela os problemas e soluções encontradas bem como algumas atividades previamente desenvolvidas que serão utilizadas nessa fase.

## 4.1 Disponibilização na Apple Store

Como citamos anteriormente, o canal de testes alfa foi disponibilizado apenas na Google Play Store, assim, nesse ponto uma das tarefas seria o suporte ao iOS, seguindo exatamente os mesmos conceitos empregados para a versão Android, com os canais de testes até que o software chegue à produção.

## 4.2 Implementação do Servidor

Citado na seção 3.2, o aplicativo contará com o consumo de dados por API e pelo servidor. Nesse segundo momento está previsto que tenha a integração com o servidor que será utilizado de suporte para o desenvolvimento da nova funcionalidade do sistema, o menu Oportunidades.

## 4.3 Menu Oportunidades

Na Engenharia de Software usamos o termo "dores"para se referir a pontos que o que está sendo desenvolvido irá solucionar em relação a quem consumirá a aplicação.

Para o DCC News, uma das maiores dores que foi notado no planejamento é a que a comunidade acadêmica não consegue de forma fácil se cadastrar em oportunidades oferecidas pelo departamento, seja ela estágios, iniciações científicas, eventos, palestras entre outros.

Assim, logo para a segunda parte do projeto, um dos focos principais do planejamento é o desenvolvimento desta funcionalidade atrelado ao desenvolvimento do servidor citado anteriormente, pois a partir dele podemos construir essa e outras funcionalidades para o aplicativo.

## 4.4 Testes

Como vimos na seção 3.4.1, tivemos desafios significativos ao longo do desenvolvimento dessa primeira parte.

Para a segunda parte do projeto, pretendemos aprimorar a qualidade dos testes disponíveis no projeto explorando o conceito de Vladimir Khorikov, [3], sobre o formato losangular dos testes.

## 4.5 Teste Beta

Quando as funcionalidades propostas para a segunda parte estiverem implementadas e testadas, partiremos para a segunda fase dos testes de aceitação: **testes beta**.

Com um número maior de usuários e já disponível em ambas as plataformas, podemos promover o aumento desses testadores do sistema e simular um ambiente o mais próximo do real possível, assim será possível refinar todos os detalhes para que o aplicativo de fato seja entregue em produção aberta a toda comunidade.

## 4.6 Aplicativo em produção

Por fim, no período final da segunda parte do projeto, quando as funcionalidades já estiverem sendo desenvolvidas e testadas, quando os testes de aceitação se mostrarem positivos e já houver a validação por parte do cliente, podemos lançar o aplicativo a produção.

# Capítulo 5

# O desenvolvimento da fase 2

Até esse momento apenas relembramos toda trajetória do aplicativo bem como os trabalhos futuros que foram previstos para essa etapa. A partir desse ponto, focaremos na etapa 2 do projeto do aplicativo do DCC, detalhando como foi o desenvolvimento dessa fase e seus desafios.

## 5.1 DCC News agora é DCC App

Na primeira fase do projeto, seguindo as ideias apresentadas acima, concebemos para o projeto o nome de DCC News, pois retrata com maior fidelidade a funcionalidade até então principal do software, as notícias.

Com o avançar do desenvolvimento, modificações e incrementos de funcionalidades foram realizadas no aplicativo alterando o requisito central do sistema, passando não só a abranger notícias mas também como um portal para a comunidade acadêmica acerca do departamento, com funcionalidades como: menu oportunidades, autenticação do usuário, ofertas de disciplinas e outras.

Dado todo esse cenário, é importante entender que o nome do produto deve representar a proposta que ele deve atingir em relação ao cliente final. Logo, o DCC News passou a ser **DCC App**, já que agora ele não conta somente com funcionalidade de feed de notícias, mas sim uma aplicação completa e pronta para atender a comunidade.

## 5.2 Projetando o menu Oportunidades

Antes de iniciar o desenvolvimento da funcionalidade principal, o menu Oportunidades, precisaríamos construir a base que seria utilizada por ela, sendo assim nessa seção

iremos detalhar os desafios, sucessos e fracassos até que a funcionalidade seja integrada a ramificação principal e disponibilizada para os usuários.

### 5.2.1 A ideia

Como foi acordado nos requisitos e na modelagem do sistema que foram brevemente descritas nesse documento e detalhadas na fase anterior, essa funcionalidade vem para resolver uma "dor"dos usuários, facilitando o acesso a inscrição em oportunidades do DCC. Relembrando, no departamento há uma divergência entre os formulários de inscrição para oportunidades, pois cada professor/coordenador e/ou responsável por esta define seus próprios parâmetros de avaliação. Logo o aluno interessado deve para cada interesse preencher um formulário específico contendo informações específicas o que pode causar um certo desconforto para encontrar documentos. Assim, esperamos resolver essa "dor"e diminuir a burocracia tanto para alunos quanto para os responsáveis pela vaga.

Essa função embarcaria um servidor de autenticação no qual o usuário se conectava com sua conta do DCC/UFMG e anexar seus documentos, histórico escolar e currículo, ele estaria habilitado a se inscrever em qualquer oportunidade que desejar. O tratamento dos dados e envio para o banco seria feito pelo próprio aplicativo garantindo a inscrição do aluno e diminuindo o tempo necessário para os preenchimentos de formulários com características específicas.

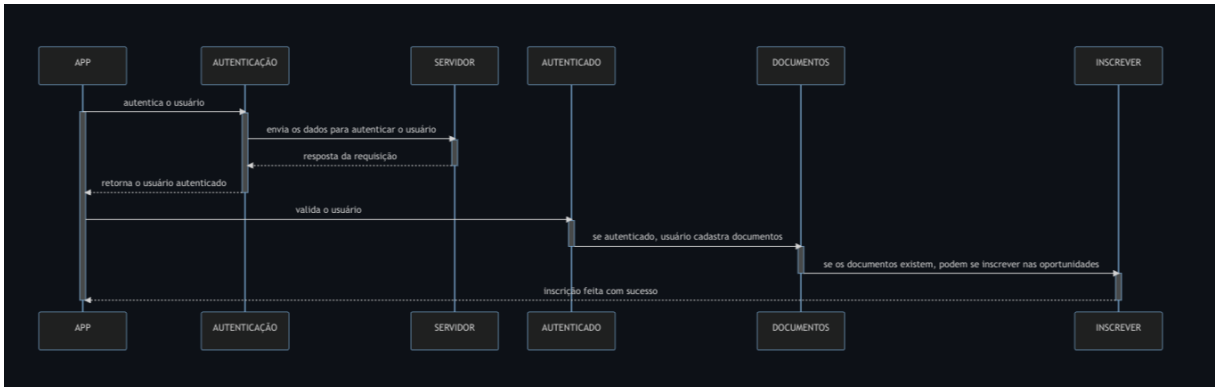



Figura 5.1: Modelagem do menu Oportunidades

### 5.2.2 O fluxo da funcionalidade

Logo, como podemos ver na figura 5.3, a modelagem dessa funcionalidade se dá por dois grandes fluxos que compreendem a autenticação do usuário e a inscrição do aluno.

- Fluxo de autenticação:

  Nessa primeira etapa é verificado se o usuário está autenticado no aplicativo, caso não esteja será solicitado que ele valide suas credenciais, usuário e senha @dcc, uma conexão com o servidor de autenticação que será explicado posteriormente é feita e caso os dados estejam corretos, uma requisição com as informações pessoais necessárias do usuário é retornada, desse modo a guia de anexo de documentos é disponibilizada para completar seu cadastro no aplicativo.

- Fluxo de inscrição:

  Nessa etapa, com o usuário já cadastrado no aplicativo, isto é, autenticado e com os documentos necessários inseridos, a opção de inscrição nas oportunidades fica disponível sem que haja a necessidade do aluno recadastrar seus dados e documentos para cada inscrição realizada, bastando realizar esse fluxo apenas uma vez, já que seus documentos ficam salvos no servidor.

Por tanto, com esse fluxo espera-se que o problema de repetição do mesmo processo para cada inscrição realizada diminua, além disso, outro ganho é a unificação do formulário de oportunidades, não sendo mais necessário para o aluno reunir documentos diferentes para cada processo de interesse.

### 5.2.3 O planejamento do servidor

Com a modelagem e o fluxo da funcionalidade definidos, estamos aptos para implementar. Seguindo o roteiro prático do livro Engenharia de Software Moderna [6], após as definições da modelagem e arquitetura utilizadas, podemos implementar a tarefa.

Como citado na seção 2, estava planejado para a aplicação o uso de uma API e de um servidor para suportar todas as funcionalidades do sistema. Na primeira fase do projeto a API via feed RSS foi desenvolvida, já na segunda etapa o foco seria o desenvolvimento do servidor utilizado para autenticar o usuário e para armazenar os documentos necessários já descritos anteriormente. Assim, foi necessário um estudo sobre

qual tecnologia poderia ser agregada ao DCC App para suportar o servidor de modo que cause o menor impacto possível no restante do sistema. Com isso, chegamos a conclusão de que o uso da tecnologia em nuvem **Firebase** e um servidor utilizando **node.js** seriam opções viáveis para o sistema de modo a causar o menor impacto possível no restante da aplicação.

### 5.2.4 O desenvolvimento dos servidores

O primeiro servidor desenvolvido após estudos foi o responsável pela autenticação dos usuários. Como estamos utilizando um framework que utiliza javascript, react-native, o uso do motor node.js é altamente recomendável pela sua fácil integração.

Descrevendo de forma detalhada, o servidor de autenticação é um container executando uma requisição ao serviço *ldap*, um protocolo padrão que fornece meios de armazenar e recuperar informações sobre pessoas, grupos ou objetos em um servidor de diretórios X. O DCC App envia uma requisição para esse servidor com as informações necessárias para a validação dos dados. De posse desses dados uma consulta a esse serviço é feita, caso os dados estejam corretos, uma requisição de retorno com os dados do aluno é enviado ao aplicativo que por sua vez realiza a autenticação do usuário.

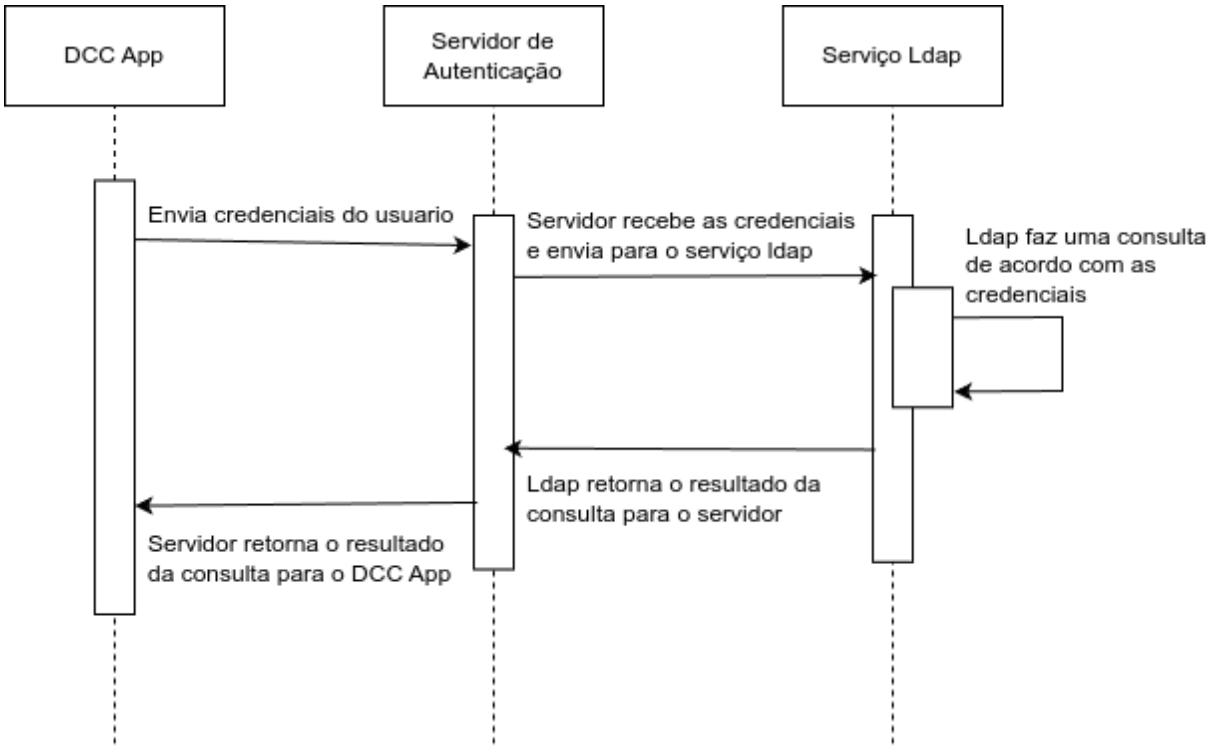



Figura 5.2: Modelagem do servidor de autenticação

Logo podemos ver com clareza o fluxo de autenticação utilizando uma técnica de modelagem da Engenharia de Software na qual criamos diagramas dinâmicos que são usados quando se pretende explicar o comportamento de um sistema, em um determinado cenário fluxo chamado de **Diagrama de Sequências**.

O segundo servidor desenvolvido utiliza a tecnologia em nuvem **Firebase**. Essa tecnologia é um conjunto de serviços de computação em nuvem de back-end e plataformas de desenvolvimento de aplicativos fornecidos pelo Google. Ele hospeda bancos de dados, serviços, autenticação e integração para uma variedade de aplicativos, incluindo Android, iOS, JavaScript, Node.js, Java, Unity, PHP e C++. Por conta do forte uso para aplicativos incluindo node.js e os sistemas Android, iOS e uma biblioteca própria em react-native para esse gerenciamento, optamos por utilizá-lo.

De forma detalhada, a hospedagem dos documentos e das inscrições se dá por meio de dois serviços do Firebase, o *Firebase Database* e o *Firebase Storage*. O primeiro serviço é relacionado a um banco de dados não relacional. Nele, armazenamos as inscrições dos alunos, documento esse composto pelos campos; nome completo do inscrito, e-mail curso, link para os documentos anexados e identificador da oportunidade. Assim, quando um usuário se candidatar a uma vaga, um documento é gerado e armazenado nesse banco de dados, ao final do processo de forma automática esse dado é integrado ao formulário de inscritos. O segundo serviço utilizado é o Storage, nele são armazenados os dois documentos necessários para as inscrições, histórico escolar e currículo. Esses arquivos são salvos em formato PDF e ficam disponíveis via link de modo que possa ser adicionado ao documento de inscrição pelo próprio sistema.

Como podemos notar, o aplicativo se comunica com dois servidores diferentes onde cada um desses tem uma responsabilidade diferente, seguindo as boas práticas da Engenharia de Software, garantindo que haja redundância em caso de falhas e removendo os acoplamentos. Além disso, seguindo o padrão de projeto **Mediador**, toda essa comunicação é feita por essa interface abstraindo do restante do projeto camadas que são necessárias somente para o tratamento desse fluxo, garantindo a legibilidade e manutenibilidade do sistema já projetadas desde a primeira fase.

### 5.2.5 A arquitetura MVC no desenvolvimento

Como citamos nas seções anteriores, o aplicativo está utilizando a arquitetura Model-View-Controller por trazer aspectos de similaridade com as definições do projeto.

Traçando um paralelo com essa arquitetura podemos evidenciar os três pontos principais do MVC para com o projeto DCC App no que tange o menu Oportunidades,

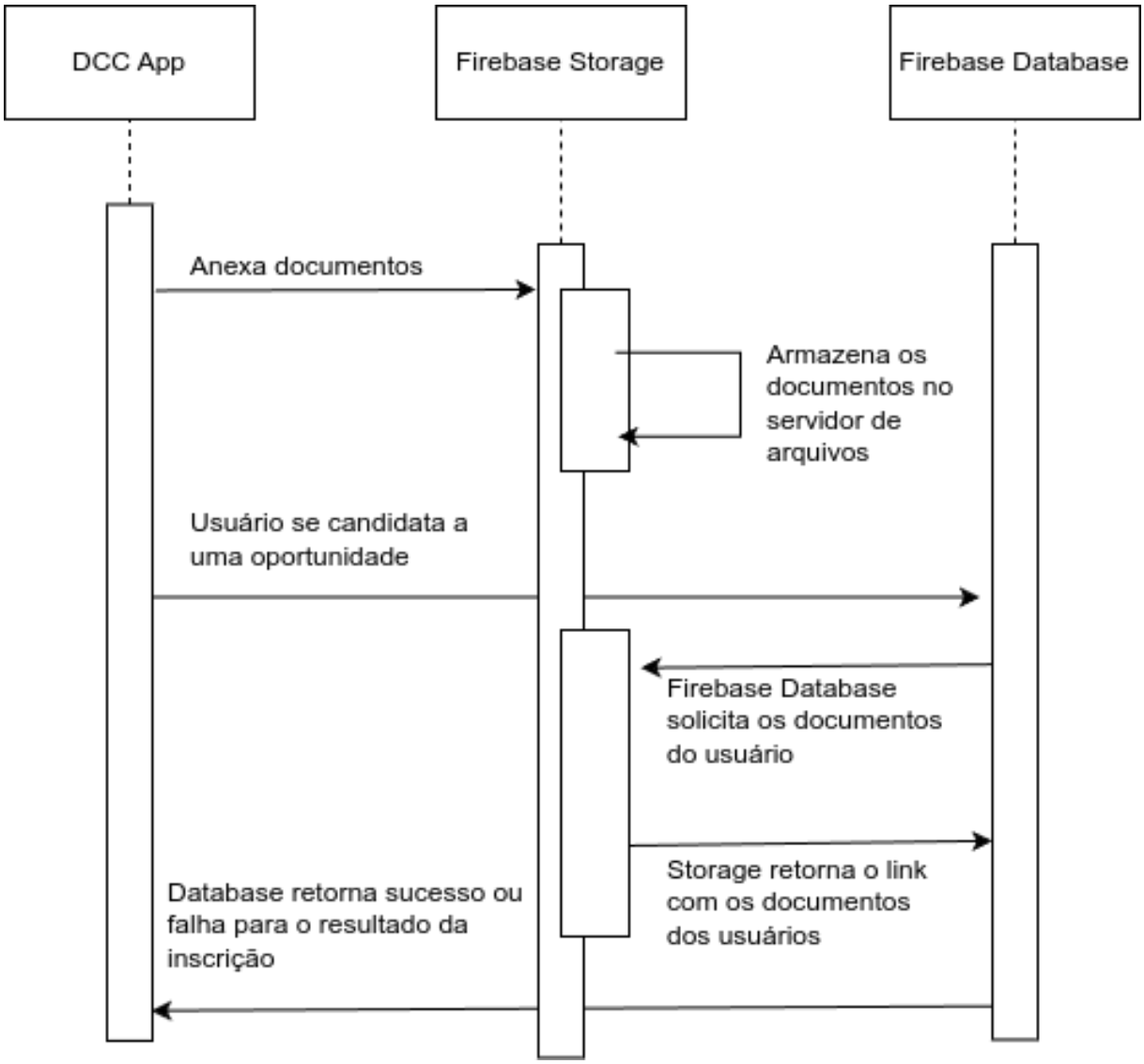



Figura 5.3: Modelagem do servidor de inscrição

bem como sua disposição em pastas no projeto.

- **View**: a visão é responsável pela apresentação gráfica do sistema.

  Para o DCC App essa parte está presente em *components* de Autenticação e de Oportunidades, local esse onde toda a interface foi implementada.

- **Controller**: o controlador é responsável por tratar os eventos gerados no sistema, por exemplo por dispositivos de entrada, teclado, mouse e assim, solicitar ao modelo ou a visão que o estado seja alterado.

  No DCC App, o controlador é definido pelo *mediator*, pois como já explicamos ele identifica as alterações de estado solicitando ao modelo os dados para que seja alterado na visualização.

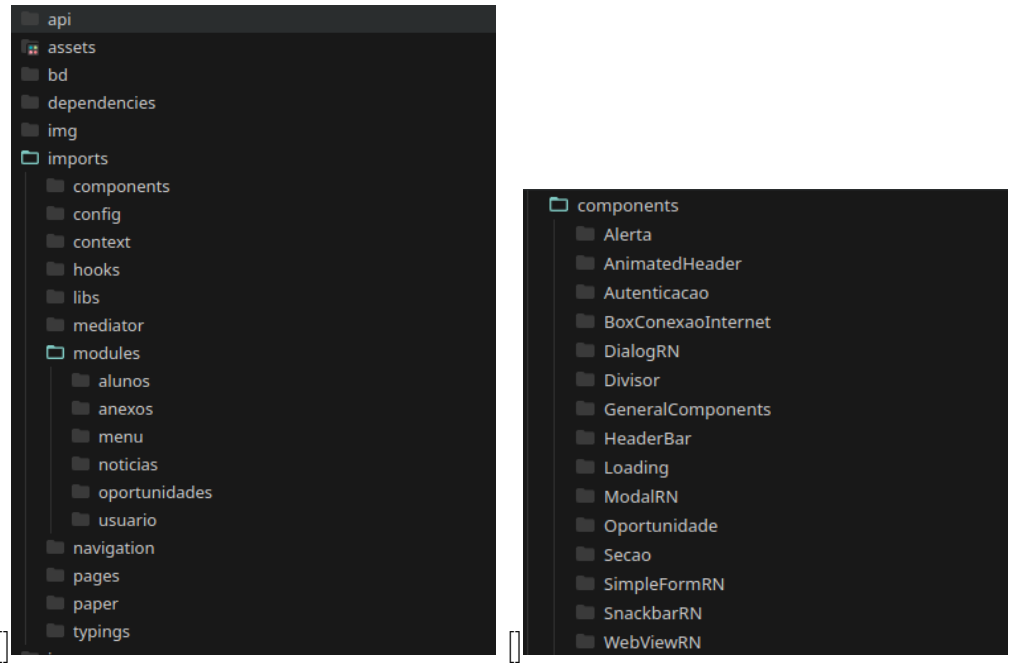



Figura 5.4: (a) Estrutura de pastas do DCC App (b) A pasta de View, com seus componentes

- **Model**: o modelo é o responsável por armazenar os dados manipulados pela aplicação e que tem a ver com o domínio do sistema. Além disso, o modelo também não tem conhecimento ou dependência sobre a visão e o controlador.

  Por fim, no aplicativo, o modelo está presente na *api* juntamente com *modules*, responsáveis pela coleta e tratamento dos dados a fim de ser disponibilizado para as outras camadas da aplicação.

### 5.2.6 A integração com o aplicativo

A escolha dessas tecnologias se deu pelo seu suporte aos componentes já utilizados no sistema. Assim, para o servidor de autenticação, a integração com node.js já existia por estarmos utilizando o framework react-native. Já o servidor de inscrição Firebase, utilizamos a biblioteca **react-native-firebase** que já possui todas as integrações necessárias para o funcionamento correto, não ocasionando nenhum impacto no restante do sistema.

É importante ressaltar que inicialmente, pretendia-se autenticar os usuários por meio das contas Google no qual o serviço do Firebase possui suporte, entretanto, durante a modelagem e estudo encontramos um limite de autenticações das contas pela própria Google, com isso optamos pela autenticação via contas DCC/UFMG.

Dado todo o exposto, detalhamos como foi o desenvolvimento desta funcionalidade, desde sua modelagem, implementação e integração com o aplicativo DCC App, de modo que essa nova feature possa agregar e modernizar os processos de inscrição em oportunidades ofertadas pelo departamento.

## 5.3 Alterações e melhorias na estrutura geral do aplicativo

Depois que a implementação do menu Oportunidades aconteceu, foi necessário repensar algumas áreas do aplicativo que já não respeitavam mais o ideal proposto. Sendo assim, houve uma mudança na iconografia e nos nomes de algumas guias e menus para que possam representar com maior precisão a sua ideia.

## 5.4 Testes

Seguindo o catálogo de boas práticas e técnicas da Engenharia de Software [6] o uso de testes se faz necessário para manter a qualidade do projeto em questão.

### 5.4.1 Github Actions

O Github Actions é uma poderosa ferramenta de integração contínua, com ela é possível a cada commit, push, ou pull executar a suíte de testes para validar as alterações. A configuração dessa ferramenta despendeu um custo significativo de desenvolvimento já que o ambiente operacional é mobile, o que envolve alguns mecanismos além do que tradicionalmente é utilizado.

Nesse workflow desenvolvido, por se tratar de um aplicativo mobile, foi necessário gerar uma cache para o*Gradle*, um sistema de automação de compilação utilizado pela Android para compilar todo o projeto e gerar um executável para o sistema operacional chamado de .apk.

```yaml
jobs:
  job-app-test:
    runs-on: ubuntu-latest
    name: DCC App CI
    defaults:
      run:
        working-directory: ./src
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Configurando ambiente node
        uses: actions/setup-node@v3
        with:
          node-version: 20.x
      - name: Instalando dependências
        run: npm install
      - name: Run Tests
        run: npm test

      - name: Cache Gradle Wrapper
        uses: actions/cache@v2
        with:
          path: ~/.gradle/wrapper
          key: ${{ runner.os }}-gradle-wrapper-${{ hashFiles('gradle/wrapper/gradle-wrapper.properties') }}

      - name: Cache Gradle Dependencias
        uses: actions/cache@v2
        with:
          path: ~/.gradle/caches
          key: ${{ runner.os }}-gradle-caches-${{ hashFiles('gradle/wrapper/gradle-wrapper.properties') }}
          restore-keys: |
            ${{ runner.os }}-gradle-caches-

      - name: Executando gradlew
        run: cd android && chmod +x ./gradlew

      - name: Build App Android
        run: |
          cd android && ./gradlew bundleRelease --no-daemon
```

Figura 5.5: Workflow do Github Actions

Com todas as configurações prontas foi possível executar a cada pull-request para a branch principal as suítes de testes, facilitando a identificação de quaisquer eventuais problemas que possam aparecer no decorrer do desenvolvimento do aplicativo.

### 5.4.2 O problema: testes e react-native

Ressaltamos na fase 1 do projeto a dificuldade do framework usado em criar testes unitários para o sistema. Isso acontece pois aqui temos o conceito de *componentes*.

Componentes no ambiente react-native são arquivos javascript carregados de código HTML, CSS e javascript. A ideia principal é que utilizando somente sintaxe javascript seja possível criar toda a estrutura necessária para a página, sendo possível utilizar HTML e CSS apenas utilizando os componentes e sem utilizar qualquer sintaxe de ambas as

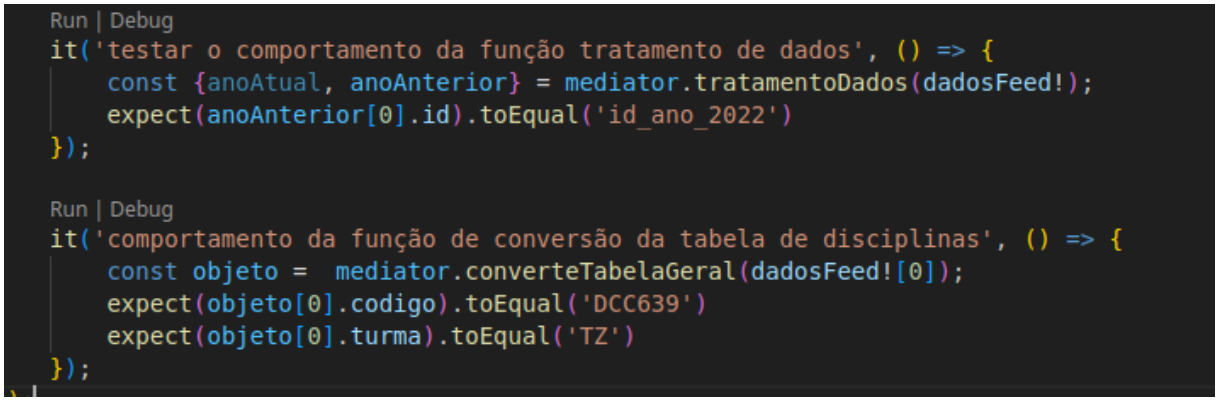
linguagens.

```tsx
import { View, useColorScheme } from "react-native"
import { alertaStyle } from "./AlertaStyle";
import Icon from 'react-native-vector-icons/MaterialCommunityIcons';
import { useTheme } from "react-native-paper";

interface IAlerta {
    detalhes: any;
    icone?: string;
}

export const Alerta = (props: IAlerta) => {
    const {detalhes, icone} = props;
    const theme = useTheme<{[key:string]: any}>();
    const { colors } = theme;
    const styles = alertaStyle(colors);
    const colorScheme = useColorScheme();

return (
    <View style={{...styles.boxAlerta,  backgroundColor: colorScheme === 'dark' ? colors.vermelhoVivoOpacoDark : colors.vermelhoVivoOpaco}}>
        <View style={styles.iconeAlerta}>
            <Icon name={icone ?? "alert-circle-outline"} size={30} color={colorScheme === 'dark' ? colors.vermelhoVivoForte: colors.vermelhoVivo}/>
        </View>
        <View style={styles.descricao}>
            {detalhes}
        </View>
    </View>
)
}
```

Figura 5.6: Formato de um código react-native

Vemos na figura 5.6, um componente react-native *Alerta* que retorna um chamado *jsx/tsx*, uma estrutura de componentes similar a árvore do DOM. Por exemplo, nessa imagem vemos o *View*, um componente react-native que possui a mesma funcionalidade de uma *div* HTML. Logo, a ideia é que esse acoplamento em componentes facilite o desenvolvimento do programador criando códigos robustos e de fácil compreensão.

Entretanto, diante do excesso de componentes dado pela própria estrutura do framework, é extremamente complexo desenvolver testes unitários para a criação de um componente já que esse envolve outros. Além disso, como componentes também são estruturas complexas na qual para cada um existe uma coleção de funções desenvolvidas, para realizar testes unitários em funções internas dessas seria necessário renderizar o próprio componente o que faria o uso de mocks e aumentaria ainda mais sua complexidade. Assim, temos testes unitários apenas para módulos independentes a exemplo do *mediator*



```tsx
Run | Debug
it('testar o comportamento da função tratamento de dados', () => {
    const {anoAtual, anoAnterior} = mediator.tratamentoDados(dadosFeed!);
    expect(anoAnterior[0].id).toEqual('id_ano_2022')
});

Run | Debug
it('comportamento da função de conversão da tabela de disciplinas', () => {
    const objeto =  mediator.converteTabelaGeral(dadosFeed![0]);
    expect(objeto[0].codigo).toEqual('DCC639')
    expect(objeto[0].turma).toEqual('TZ')
});
```

Figura 5.7: Dois testes dentro da suíte para a pasta mediator, presente na figura 6

### 5.4.3 O problema dos testes de integração com os componentes e testes manuais

Como abordamos na fase 1 do projeto, a partir de uma análise exploratória acerca da literatura chegamos as definições de Vladimir Khorikov expostas na seção 3.4.1 na qual planejamos para essa fase o desenvolvimento das suítes em um formato losangular focado nos testes de integração.

Novamente, para desenvolver os testes de integração encontramos o problema dos componentes. Por acoplarem o código necessário para que aquele componente funcione, para testar algum método dele, seria necessário testar o componente em si, o que não é interessante e viável atualmente. Sendo assim, optamos por preservar a integridade dos componentes e para todos aquelas classes e funções que não havia nenhum tipo de conexão com quaisquer componentes, realizamos os testes dentro das suítes como exposto na figura 5.7. Já para aqueles que não foram cobertos, assim como na fase 1 do projeto, realizamos testes manuais o que cai em desencontro com as boas práticas e técnicas da Engenharia de Software, mas resolve o problema da falta de cobertura para tais trechos do projeto.

## 5.5 Testes Beta

Nesse ponto alcançamos a fase final de desenvolvimento seguindo as boas práticas e técnicas da Engenharia de Software.

Nesse último aspecto estamos iniciando a fase conhecida como **teste de aceitação**. Relembrando, testes de aceitação são aqueles que servem para verificar se o sistema que foi desenvolvido está em concordância com as necessidades dos clientes. Eles possuem duas características: são testes manuais feitos pelos clientes do produto e também envolve um processo de validação do sistema, logo, caso tudo esteja de acordo, o produto entra para produção.

Esses testes podem ser divididos em duas fases: testes alfa e testes beta. A primeira fase foi concluída na primeira parte do projeto em um ambiente controlado e com acompanhamento dos desenvolvedores, como define a técnica. Nessa segunda parte do projeto, lançamos a próxima fase dos testes de aceitação, **testes beta**, nessa fase não temos mais um ambiente controlado, o grupo de usuários é maior e há acompanhamento dos desenvolvedores. Para isso, movemos nossa aplicação para o ambiente de testes aberto

do Google Play Console, gestor do aplicativo na loja Play Store, limitando o número de usuários em 150. Logo com um aumento de 16 para 150 garantindo que mais usuários possam utilizar o software, gerar feedbacks e promover a aceitação para a comunidade.

Diante do exposto, objetivamos mais uma etapa do desenvolvimento seguindo as boas práticas e técnicas da Engenharia de Software, partindo para a fase final.

## 5.6 Último passo: aplicativo em produção

A última etapa do desenvolvimento de software seguindo os conceitos da área e também a última etapa desse projeto é o chamado *"colocar em produção"*. Esse termo na área significa que um produto está pronto para ser comercializado amplamente, pois nesse projeto e como segue os conceitos da área já percorreu por todas as fases do desenvolvimento.

O DCC App, com isso consegue alcançar todo o fluxo bem definido seguindo o roteiro prático do livro Engenharia de Software Moderna [6], evidenciando a importância dos conceitos oferecidos pela área para desenvolver um software de qualidade para o mercado.

# Capítulo 6

# O problema: Apple Store e UFMG

Toda a documentação exposta na seção 4, tomou-se como verdade para os ambientes operacionais Android e iOS, entretanto, a única etapa que estava prevista para a fase 2 do projeto 3 e não foi executada é a *Disponibilização na Apple Store*.

Apesar de termos em mão um produto funcional para ambos sistemas operacionais já testados em ambientes virtuais, a disponibilização na loja oficial da Apple até o presente momento foi adiada. Esse adiamento se deu por questões burocráticas impostas pela Diretoria de Tecnologia da Informação (DTI) da UFMG. Sendo assim, os testes de validação e aceitação até o presente momento contam apenas com experiências de uso no sistema Android.

Com isso, a disponibilização do aplicativo para produção também será restrita a esse sistema operacional até que as questões burocráticas sejam resolvidas com a universidade. Logo, ao ser sanado, retomaremos os passos propostos pela Engenharia de Software com os testes de aceitação e validação por parte dos usuários de iOS até que o mesmo também seja disponibilizado amplamente.

# Capítulo 7

# Conclusão

Durante toda essa documentação, expomos o passo a passo para que o projeto seja guiado seguindo as boas práticas e técnicas da Engenharia de Software no seu maior rigor possível.

É importante dizer que a Engenharia de Software possui uma gama vasta de conceitos que podem ou não ser aplicados a depender do escopo do produto. Nesse projeto desde sua primeira fase, com a definição de tecnologia, modelagem, arquitetura, requisitos do sistema, até a entrega final do produto, percorremos por diversos conceitos que são amplamente abordados em grandes empresas e áreas da tecnologia visando facilitar o desenvolvimento de um software.

Para essa fase 2 reforçamos a conclusão alcançada na fase 1, de o quão vantajoso é para o desenvolvimento de um projeto utilizar as boas práticas e técnicas de Engenharia de Software. A exemplo disso, podemos destacar o desenvolvimento dos servidores de autenticação e inscrição que já haviam sido contemplados nos requisitos do sistema no início do projeto, o que deu condições para que o desenvolvimento do sistema esteja atrelado ao suporte aos incrementos de funcionalidades no futuro garantindo que o código continue seguindo os padrões arquiteturais e de projeto definidos inicialmente. Assim, com todo esse roteiro traçado podemos contornar problemas que sem eles poderiam ser grandes complicadores para o bom andamento do projeto.

É importante também destacar a experiência no desenvolvimento mobile. Apesar desse mundo mobile estar em voga nos tempos atuais, o meio carece de frameworks capazes de atender os dois ambientes, Android e iOS, de forma unificada utilizando o mesmo código. Uma solução também famosa e bastante utilizada é o *Flutter*, um framework multiplataforma assim como o react-native desenvolvido pela Google. Cabe enfatizar aqui que esses dois serviços possuem os mesmos problemas, já que com a tecnologia atual é extremamente difícil conseguir desenvolver um framework capaz de atender dois ambientes completamente diferentes utilizando a mesma codificação.

Outra questão que pode ser levantada está em torno das diferenças para o desenvolvimento web. Enquanto na web a criação do setup de desenvolvimento, bibliotecas e comunicação é muito mais simples pois nela estamos lidando diretamente com navegador. Para o ambiente mobile, esses aspectos ficam restritos. Em primeiro lugar, o setup de desenvolvimento é mais complicado, pois é necessário um emulador android, para o Android,

é um aplicativo chamado *xcode* disponível somente no computador da Apple, ou seja, para o desenvolvimento do iOS é necessário um ambiente com um sistema operacional macOS. Em segundo lugar, o arcabouço de bibliotecas é escasso pois várias delas já foram descontinuadas ou não oferecem suporte para as versões mais novas do Android e do iOS. Assim, apesar dessas diferenças e algumas questões mais complexas, o desenvolvimento mobile utilizando o código nativo de cada sistema operacional abre possibilidades de exploração maior do que no desenvolvimento web, já que não há a camada do navegador atuando diretamente no sistema, o que torna seu aplicativo mobile mais integrado e próximo ao sistema operacional.

Para esse projeto um dos maiores problemas enfrentados foi a criação dos testes. O uso de testes manuais não é uma prática totalmente repreendida, porém em relação a literatura deve ser evitada ao máximo. Para trabalhos futuros esperamos estudar mais sobre o comportamento do framework em relação a testabilidade dos componentes e funções, aumentando a sua qualidade e garantindo uma cobertura maior do sistema.

Portanto, assim como na fase 1, vimos também que apesar de seguir rigorosamente os conceitos, alguns pormenores ainda podem acontecer pois em cenários reais, temos um ambiente que não é controlado e assim casos expoentes surgem como as questões envolvendo a disponibilização na Apple Store. Porém percebemos que com o devido planejamento e organização essas questões acabam sendo menos impactantes no projeto, logo avaliamos que para que um projeto seja bem desenvolvido, ter o conhecimento sobre todo conceito proposto pela literatura é fundamental para que seu produto se torne capaz de atender as necessidades para a qual ele foi proposto.

# Referências Bibliográficas

[1] *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., USA, 1999.

[2] Johnson Ralph Vlissides John Gamma Erich, Helm Richard. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-wesley professional edition, 1994.

[3] Vladimir Khorikov. *Unit Testing Principles, Practices, and Patterns.* Manning edition, 2020.

[4] Arnon Sturm Mira Balaban. Software engineering lab – an essential component of a software engineering curriculum. 2018.

[5] Ken Schwaber. Scrum development process. In Jeff Sutherland, Cory Casanave, Joaquin Miller, Philip Patel, and Glenn Hollowell, editors, *Business Object Design and Implementation*, pages 117–134, London, 1997. Springer London.

[6] Marco Tulio Valente. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade.* Independente edition, 2020.