Desenvolvimento de um aplicativo móvel para gestão de doações para entidades filantrópicas

Felipe Toshio Amanuma Soares[1], Hilton Marins[2], Ricardo Santos[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - *Campus* Juiz de Fora

[2]Núcleo de Informática - Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - *Campus* Juiz de Fora

felipetoshio.contato@gmail.com, hilton.junior@ifsudestemg.edu.br, ricardo.santos@ifsudestemg.edu.br

Abstract. *The aim of this article is to show a mobile application with the main feature of managing attributes and needs in the donations for philanthropic entities in the city and region, bringing visibility to them and safety for the donator as well. From search and circumscription of this project were chosen the following tools: the development stack of Microsoft based in the back-end and DevOps, and for the front-end It's used react-native, the Facebook's team created library.*

Resumo. *Este artigo possui o objetivo de demonstrar um projeto de um aplicativo móvel que terá como funcionalidade principal gerenciar as características e as necessidades das doações feitas às instituições filantrópicas da cidade e região, além disso proporcionará maior visibilidade e segurança para seus doadores. Durante a pesquisa e formatação do projeto foram elencadas as seguintes ferramentas: o stack de desenvolvimento da Microsoft como base do back-end e DevOps, já o front-end baseado em react-native, biblioteca criada pelo time do Facebook.*

1. Introdução

A tecnologia tem se apresentado no cotidiano como um artigo de subsistência e, durante este período pandêmico, se fez mais urgente do que nunca, pois foi nesse tempo de enclausuramento que os desafios se tornaram maiores a cada dia. O que era uma simples ida ao mercado, se tornou algo quase inviável e exaustivo psicologicamente.

Essa evolução no modo como percebíamos a tecnologia, transformou drasticamente nosso modo de vida: aulas sendo feitas via internet, atendimento ao cliente sendo realizado *online* e até nossas compras realizadas da mesma forma, em certos casos, sem a necessidade de intervenção humana.

Porém essa nova forma de vida e interação social trouxeram consequências expressivas para uma camada da sociedade que se tornou invisível, seja pelas características econômicas ou seja pela ausência da própria tecnologia. Diante do processo de "afastamento" a que esta camada social foi conduzida, levou as instituições filantrópicas a ampliarem seu escopo de trabalho.

Projetos como o Anjos da Rua[1], que visa a distribuição de roupas e alimentos

[1] O coletivo Anjos da Rua, pode ser encontrada pelo endereço do *Instagram*: @anjosdaruajf

à pessoas em situação de rua e a ABAN [2], uma ONG que atua desde 1997 em Juiz de Fora com vários projetos que buscam minimizar a miséria e a injustiça social por meio de oficinas de arte, esporte e educação para que os indivíduos se tornem ativos dentro da comunidade em que se encontram. O projeto também promove eventos que arrecadam doações para complementar o auxílio às famílias dos seus assistidos.

Em entrevista com diretores e líderes destes projetos, constatou-se um cenário de recessão econômica anterior à própria pandemia. Os níveis de doações encontravam-se baixíssimos, com uma leve reação durante o início da quarentena, diante da dificuldade relatada pelos entrevistados em atender todas as famílias assistidas.

Existem muitos fatores que contribuem para as instituições estarem neste estado, dentre eles, o mais impactante além do fator econômico, é a falta de visibilidade nas mídias sociais. E, mesmo quando são reconhecidas pela sociedade, muitas das vezes, não atingem o público apto a auxiliar que não o fazem, na maioria das vezes, por não conhecerem ou ter acesso de forma segura aos projetos

Outros pontos críticos para o gestor das instituições, são a administração e o voluntariado. Existem demandas próprias que as envolvem, como a criação de eventos e projetos que, muitas vezes, não acontecem por ausência de uma divulgação efetiva, e uma ferramenta para convidá-los a participar. A saída é buscar contato via *whatsapp*, pedindo pelo auxílio sempre que necessário. Novamente, destaca-se a necessidade de visibilidade que as instituições buscam e demonstrar o quão sério e necessário é seu trabalho.

O fato é que, diante das falas, fica evidente a distância entre o possível público doador e as instituições, pautados, principalmente, pela insegurança que nasce a partir da divulgação de instituições que, por desorganização ou descontrole, não conseguem demonstrar os frutos das doações recebidas.

A partir dos relatos, percebeu-se a necessidade da criação de um projeto que pudesse levar maior confiança ao público-alvo doador e visibilidade às instituições.

A forma definida para a possibilidade de solução deste problema foi de criar um aplicativo móvel para controle e divulgação de doações para as entidades filantrópicas certificadas pela sociedade e autoridades locais. São entidades que possuem documentação fornecida pelos diversos órgãos públicos ligados às áreas sociais e filantrópicas do município. Este projeto visa trazer praticidade, facilidade e seriedade para o público que tenha interesse em participar por meio de doações às instituições filantrópicas.

2. Metodologia

Durante o processo de busca por literaturas que refletissem de forma mais assertiva todo o processo de gestão de entidades filantrópicas, constatou-se que não traziam para nós a resposta para a problemática aqui exposta.

Artigos como o do autor Roger A. Lohmann: "*Charity, philanthropy, public service, or enterprise: what are the big questions of nonprofit management today?*", que explicita bem o funcionamento de instituições públicas e não-públicas de caridade,

[2] A ONG ABAN (Associação Dos Amigos), localizada na Rua João Beguelli,72, Bairro Dom Bosco, Juiz de Fora. Também pode ser encontrada nas redes sociais pelo endereço do *Instagram*: @abanjf

acabam possuindo um enfoque crítico na forma como o trabalho vem sendo feito e mostram uma realidade diferente da vivida regionalmente, que é o foco deste trabalho.

Para solucionar este problema, foram feitas sessões de entrevistas com líderes de instituições locais que poderiam sanar dúvidas e trazer informações. Além disso, explicitar suas maiores dificuldades diárias para que, assim, fosse realizada alguma melhoria para a qualidade de gestão destas instituições que, por sua vez, responderão a sociedade como um todo.

As entrevistas foram todas feitas por meio de plataformas de comunicação online, por conta do distanciamento social, e focadas, principalmente, em duas questões: como o momento afetou os processos da entidade e como o aplicativo poderia ajudar nas questões cotidianas da instituição.

Com as informações adquiridas, foi possível montar os requisitos funcionais e ter uma melhor ideia das reais necessidades das instituições. Também foi possível criar regras de negócio de forma mais assertiva. Exemplificando, temos a regra de cadastro para entidades filantrópicas, ponto fortemente discutido por conta da dificuldade de atender instituições menores com segurança para o doador.

Sendo assim, optamos realizar uma pesquisa qualitativa, buscando entender melhor as nuances explicitadas, não se atentando somente a números e sim a uma interpretação dos dados apresentados pelos entrevistados.

Quanto à natureza da pesquisa, esta sendo do tipo aplicada, gerou uma hipótese de solução prática, a partir dos dados coletados, que tentará de alguma forma minimizar o problema. Os procedimentos usados para levantamento destes dados se apresentaram a partir da pesquisa de campo e experimental.

2.1. Dos requisitos funcionais

Após a coleta de dados, compilados a partir das informações prestadas pelas entidades pré-selecionadas, compreendemos que o aplicativo móvel poderia trazer para as mesmas, requisitos funcionais que poderiam ser aplicados para um *MVP*[3], que são os requisitos mínimos de um aplicativo pronto para uso inicial do usuário.

Também foram compreendidos outros requisitos que poderiam ser considerados *stretch,* como por exemplo, gerenciamento de voluntários,ou seja, implementações futuras, mas que serão consideradas na sequência de utilização pelo usuário, conforme as entregas do produto.

Os requisitos funcionais definidos foram:

Tabela 1. Definições dos requisitos funcionais

Requisito Funcional	Descrição
Autenticação (RF01)	O sistema contará com a página de autenticação onde qualquer tipo de usuário poderá se cadastrar e entrar no sistema. A página terá a possibilidade de recuperar senha. A autenticação deve ser feita via email e senha.

[3] *Minimum Viable Product*, ou em português, Produto Minimamente Viável

	[Stretch] O sistema terá autenticação via Google e Facebook via *OAuth,* por não existir outra forma de se conectar nos dois serviços sem utilizar esta ferramenta. Desta forma, facilitando a entrada e cadastro no sistema.
Cadastrar doador (RF02)	O sistema fornecerá duas formas de se cadastrar como doador: sendo a primeira delas, através da tela de cadastro que poderá ser acessada via tela de autenticação. Nesta tela, o usuário deve inserir seus dados nos campos obrigatórios. De outra forma, somente será possível acesso como usuário administrador e, o link para entrada, só aparecerá para este mesmo tipo de usuário Somente o usuário administrador terá a possibilidade de excluir usuários do tipo doador. Sendo que, ao usuário doador, somente será possibilitada alterar o próprio usuário enquanto o administrador terá todo controle de alteração.
Cadastrar entidade filantrópica (RF03)	O sistema oferecerá duas formas de se cadastrar a entidade filantrópica. A primeira delas, através da tela de cadastro, pode ser acessada via tela de autenticação. Nesta tela, o usuário deverá inserir seus próprios dados. De outra forma, somente será possível acesso como usuário administrador e o link para entrada, só aparecerá para este mesmo tipo de usuário. Para realizar o cadastro de entidade filantrópica, o usuário deverá passar por um sistema de autorização que consiste em 3 possibilidades: 1- Ter o registro na prefeitura do município (critério a definir de acordo com a legislação em vigor) 2- Envio de documentos que certifiquem a diretoria ativa e o seu registro em cartório de uso. 3- Avaliar a possibilidade junto ao administrador do sistema para a visita *in loco* da entidade, confirmando e validando sua atividade. Somente o usuário administrador terá a possibilidade de excluir usuários tipo entidade filantrópica. Ao usuário tipado somente será possibilitado alterar a si próprio.
Cadastrar Doações (RF04)	O sistema deve permitir ao usuário tipado como entidade filantrópica criar, alterar, deletar e visualizar suas doações. Entende-se como doações objetos diversos bem como: produtos alimentícios, de higiene, educacional e financeiro que são constantes das doações entregues às instituições.

	O sistema deve permitir ao usuário doador visualizar todas as doações criadas por qualquer entidade filantrópica de seu interesse, podendo filtrar por entidade, tipo de doação, data de criação, dentre outros. Reafirmando que o usuário doador não deverá criar, deletar ou alterar doações.
Cadastrar Projetos (RF05)	O sistema deve permitir ao usuário do tipo entidade filantrópica criar, alterar, deletar e visualizar projetos. Este deve ser adicionado como opção para o usuário do sistema. Entende-se como projetos sazonais, a doação de objetos diversos, produtos alimentícios, de higiene, educacional e financeiro que tem tempo delimitado e deverão ter destaque na tela inicial do aplicativo. Um exemplo seria: no período inicial do ano letivo, a entidade faz campanha de doação de material escolar para crianças carentes. O sistema deve permitir ao usuário doador visualizar todos os projetos criados por qualquer entidade filantrópica, porém não deve permitir criar, deletar, ou alterar projetos.
Realizar Doação (RF06)	O sistema deverá mostrar opções de doação em dois locais distintos. Um deles dentro do perfil da entidade filantrópica onde ficaria amostra todas as doações e projetos ativos. O outro seria apresentado na tela inicial do aplicativo onde todas as entidades e doações poderão ser visualizadas, por ordem do mais novo para o mais antigo. Ao clicar na doação ou projeto, o sistema deve começar a iniciar o processo de doação para entidade filantrópica.
[Stretch] Gerenciar Voluntários (RF07)	O sistema deve apresentar formas de o usuário do tipo entidade filantrópica aceitar candidatos a voluntários do programa. Deve, também, mostrar uma forma de remover aqueles que já não fazem parte de seu quadro. Voluntários só podem ser usuários do tipo doador. O sistema deve permitir a entidade filantrópica estar disponível para novos voluntários ou não.
[Stretch] Candidatar a Voluntário (RF08)	O sistema deve apresentar a opção de se candidatar a voluntário de uma entidade filantrópica ao acessar o perfil da entidade. Esta opção somente poderá ser acessada por usuários doadores. O usuário do tipo entidade filantrópica receberá o pedido de voluntariado e deverá aceitar ou não a sua candidatura. Será possível no próprio responder a

	aceitação ou recusa dessa candidatura.
[Stretch] Cadastrar Evento (RF09)	O sistema deverá permitir acesso à uma tela de agenda para que o usuário seja tipado como entidade filantrópica na qual ele poderá criar, alterar, excluir e visualizar eventos. Esta tela deve permitir escolher o local, data e hora de início e fim do evento, bem como, interligá-los a um projeto e a quantidade de voluntários. O sistema deve notificar os usuários envolvidos no evento sobre o mesmo de acordo com configuração pré estabelecida.
[Stretch] Marcar data em Agenda para visitação (RF10)	O sistema permitirá ao usuário doador realizar um pedido de visitação *in loco* à sede da entidade filantrópica. Para isso, ele terá que checar a agenda da entidade e enviar um pedido constando horário de entrada e saída de acordo com a disponibilidade da entidade filantrópica. A entidade filantrópica poderá aceitar ou recusar o pedido de visitação, bem como sugerir uma nova data. Em caso de aceite, este horário será marcado na agenda do aplicativo. O sistema deve notificar ambos usuários quando faltar um dia, uma hora e no horário da visitação.

2.2. Da prototipagem

Para o desenvolvimento do aplicativo, inicialmente, foi necessário realizar uma documentação sobre a aplicação, como determina a boa prática de um projeto. Envolve criar alguns protótipos de telas e modelagens que virão a ser utilizadas na sua aplicação. Para criação destes modelos, foi utilizado pelo pesquisador o *LucidChart*, uma plataforma intuitiva que dará suporte aos fluxos de um sistema com requisitos funcionais que irão criar a interação usuário-máquina.

Esta plataforma apresenta uma ampla coleção de diagramas diferentes (conforme figura 1) e sua escolha foi idealizada pela praticidade de sua utilização, visando o melhor entendimento do projeto.

Também foi considerado na escolha do *LucidChart* como *software* padrão o fato dele possuir suporte e componentes específicos do *stack* de desenvolvimento da *Microsoft*.

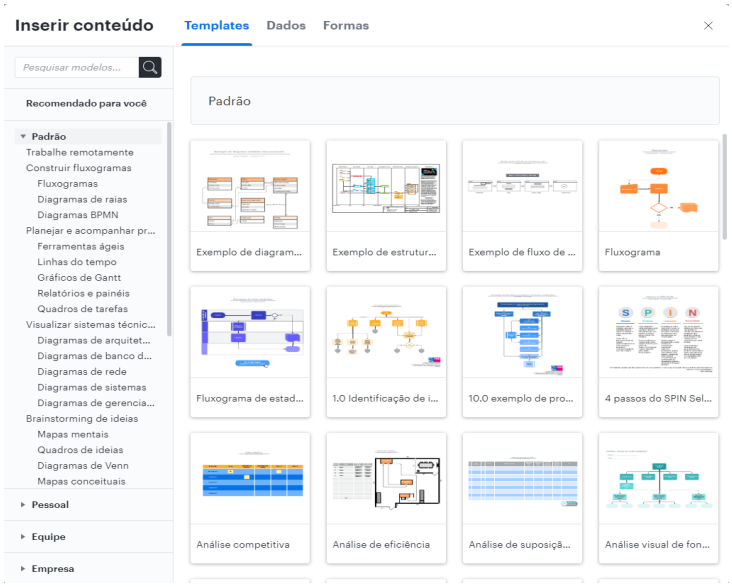

Figura 1. Página de escolha de *templates* para criação de diagramas no LucidChart. Fonte: O autor.

Outra plataforma escolhida para especificação do aplicativo foi o *Figma*. O objetivo desta aplicação foi de criar protótipos de tela de forma a caracterizar o seu *design* para demarcar especificamente o local de cada componente.

É considerado de extrema importância, nesta etapa, a especificação e criação de protótipos de telas, sendo necessário a originalidade de cada tela para posterior discussão do que será desenvolvido e de sua funcionalidade, promovendo as necessárias mudanças.

Através de pesquisas realizadas, o *Figma* foi escolhido pela sua facilidade de uso e por ser uma ferramenta gratuita, o que faz dela uma ferramenta funcional, podendo ser utilizada por qualquer usuário e executada em qualquer máquina. Outro fator relevante deste é conter uma documentação extensa sobre suas funcionalidades, dicas de *UX*[4] e tutoriais que demonstram seu funcionamento na prática. Além disso, possui uma comunidade ativa que disponibiliza componentes comuns com o objetivo de facilitar a criação de protótipos.

2.3. Do *back-end*

Após a definição de documentação desenvolvida para construção do aplicativo, foi definido o *stack* de linguagens e *frameworks* que seriam utilizadas.

Para iniciar o desenvolvimento, se fez necessária a definição da arquitetura seguida para API[5], que é um conjunto de regras que definem como dispositivos irão se comunicar entre si. Este conjunto de regras que o aplicativo deverá seguir, para se comunicar com a camada de lógica de programação, o *back-end*[6], é considerado o modelo mais seguro por possuir uma camada a mais de autenticação do que somente a de aplicação. É por este meio que se pretende acessar banco de dados que contém informações sensíveis. Desta forma, o desenvolvimento se torna menos acoplado e mais independente.

---

[4] UX - User Experience, em tradução literal, experiência de usuário

[5] *Application Programming Interface* ou Interface de Programação da Aplicação

[6] Processo interno que envolve uma aplicação.

Na possibilidade de escolha de uma modelagem de *API*, após pesquisa e comparação, a melhor apresentada foi a *REST*[7], que é um conjunto de restrições arquitetônicas que formatam a interação com a *API*.

Nesta metodologia, por conta do objetivo que a aplicação tem, as *APIs REST* possuem melhor performance em sua utilização em pequena escala e são mais flexíveis. Se adaptam a vários contextos diferentes e com comprovada eficiência devido à utilização de *JSON*[8], que são arquivos mais leves do que por exemplo *XML*[9], que é utilizado em *APIs SOAP*[10], outra arquitetura de software muito utilizada. "*Além de sua escalabilidade ser amplamente facilitada em caso de necessidade de crescimento para projetos maiores"*[11] como comprovado por Erenis Ramadani e Festim Halili (2018) no artigo *Web Services: A Comparison of Soap and Rest Services*.

Considerou-se também, que o suporte para utilização de *APIs REST* é simplificado no *.net framework*, escolhido para o desenvolvimento desta *API*, como será tratado a seguir.

Para o desenvolvimento do *back-end*, o caminho seguido foi a utilização do *.net framework*, reconhecido pela comunidade de desenvolvedores como o mais confiável *framework* para a linguagem de programação C#. Além de ser o *framework* indicado no livro *C# Essentials,* dos autores Ben Albahari, Peter Drayton e Brad Merrill (2002). A escolha dessa linguagem foi definida por ser contemporânea e com constantes possibilidades de atualização. Por possuir, também, um suporte muito bem embasado para criações de *API* por meio do *Visual Studio* que foi a *IDE* [12] eleita.

Ao criar o projeto por meio do template já existente, este *framework* vem configurado com uma ferramenta útil para o desenvolvimento de qualquer *API* chamada *swagger*. A partir dela, é possível validar exatamente os *endpoints* criados, verificando não somente sua resposta como também os *headers* enviados, bem como o objeto com a tipagem de cada campo enviado. Além do exposto, ele separa o tipo de chamada que será feita, como também, qual seria o seu caminho, como demonstrado na Figura 2.

---

[7] REST - *Representational State Transfer*

[8] JSON - *JavaScript Object Notation* ou Notação de Objetos JavaScript

[9] XML - *eXtensible Markup Language*

[10] SOAP - *Simple Object Access Protocol* ou Protocolo Simples de Acesso a Objetos

[11] Livre tradução do autor

[12] IDE - *Integrated Development Environment,* na tradução literal, Ambiente Integral de Desenvolvimento

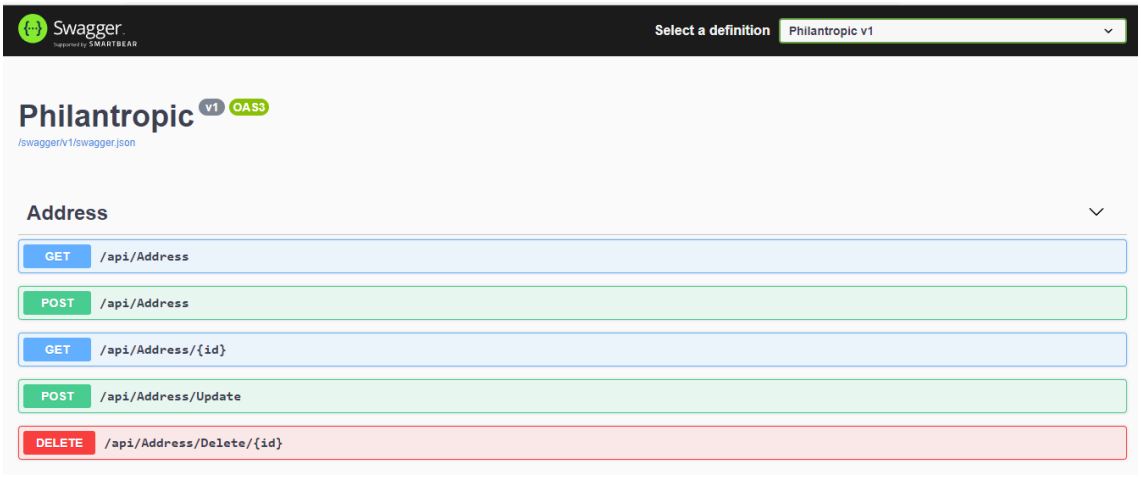

Figura 2. Página inicial do *swagger* com um exemplo dos *endpoints* de endereços. Fonte: O autor.

Outro fator que foi pesquisado, ao iniciar o desenvolvimento da *API*, foi a versão do *framework* que iria ser utilizada. E como melhor ferramenta para este projeto, a escolha foi pela versão 5.0, mesmo tendo como comparativo sua versão 6.0, que apesar de sua performance foi descartada por conta da sua instabilidade.

2.3. Dos padrões de projeto

Após a criação do projeto, os passos seguintes foram o desenvolvimento do código e da estruturação das pastas. Durante o desenvolvimento, alguns padrões de projeto e de *design* foram utilizados de acordo com a necessidade apresentada, como por exemplo o *DTO Pattern*[13] que é uma forma de carregar dados entre processos, diminuindo a quantidade de chamadas de método e encapsulando dados de forma que o usuário final não tenha acesso a todos os dados que estão sendo salvos, expondo somente o necessário, dando uma segurança extra ao projeto, semelhante ao uso explicitado por autor Floyd Marinescu (2002), que em seu artigo, além de explicar o uso do padrão, demonstra como utilizá-lo na linguagem de programação *Java*.

Dentro do *framework* utilizado, é possível, também, utilizar um padrão de projeto muito comum em sistemas, o *Factory Method Pattern*. Este padrão tem como objetivo principal reduzir o acoplamento, criando dependências flexíveis e, desta forma, através de interfaces, criar objetos além de possibilitar com que as subclasses decidam como e qual classe será instanciada no projeto da *API*. *"O Factory é muito apropriado para uso quando há necessidade de chamar determinada classe de construção parecida, que facilmente pode ser diferenciada pelo construtor"*[14], como dito pelo autor Matús Chochlík (2016) em seu artigo *Implementing the Factory Pattern with the help of reflection*.

Dentro deste trabalho, o *Factory* é utilizado para trazer uma instância do banco para dentro do projeto, disponibilizando o banco de dados. Por sua aplicabilidade, por ser um facilitador do processo e por não ter a necessidade de se iniciar sua conexão toda vez que nova operação for acontecer, fica compreendido que sua utilização neste projeto é definitiva. Dentro do *startup* do projeto ele define a conexão como demonstrado na Figura 3.

---

[13] DTO Pattern - *Data Transfer Object Pattern*, ou padrão de Objeto de Transferência de Dados

[14] Livre tradução do autor

```
services.AddDbContextFactory<ApiDbContext>(option => option.UseSqlServer(Configuration.GetConnectionString("ServerConnection")));
```

Figura 3. Adição do *Factory* através do serviço nativo do *.net framework*.
Fonte: O autor.

Em relação a padrões de *design*, uma das formas desenvolvida é o *Repository Pattern*, utilizado geralmente por dois propósitos: criar uma camada de abstração para o acesso ao banco e centralizar todas as operações que envolvem dados, além de ser uma ótima resposta para a utilização de outra *pattern*, a *DAO* [15], que em muitos casos se torna uma tarefa repetitiva e ineficiente. Além do que, quando implementado, costuma trazer o acesso ao banco para dentro da camada de serviço. Outro fator decisório foi que o padrão de repositórios encaixa muito bem com o *Factory*, apontado anteriormente, por conta de centralizar a sua chamada e com o próximo padrão que será abordado, o *Dependency Injection Pattern*.

De todos os princípios definidos por Robert C. Martin[16] (também muito conhecido como "Tio Bob"), o *SOLID* é o acrónimo mais conhecido, onde o S - *Single-responsibility Principle*, O - *Open-closed Principle*, L - *Liskov Substitution Principle*, I - *Interface Segregation Principle* e por último o D - *Dependency Inversion Principle* forma uma referência única nas boas práticas de programação. Tendo esta última letra significativa influência em um dos padrões que utilizaremos nesta *API*.

Parafraseando o pesquisador Robert C. Martin este princípio define, "*os módulos de mais alto nível em um sistema não devem depender de submódulos ou módulos de nível inferior, ambos devem depender de abstrações e estas não devem depender do detalhamento, o detalhamento que deve depender das abstrações*"[17] - Robert C. Martin e Micah Martin (2006), compreendendo a utilização do padrão de projetos Injeção de Dependência em que este princípio é respeitado.

Este *Pattern* segue exatamente o que o princípio indica. Ele adaptará de uma forma em que todas as dependências de uma classe serão carregadas no momento que forem necessárias, fazendo que a partir do contrato pré existente entre uma classe e sua interface informe o que poderá ser utilizado sem necessariamente carregar sua implementação, assim desacoplando as duas dependências.

Muitos desenvolvedores defendem a utilização desse padrão devido à sua importância. Dentre os motivos apresentados, aprimorar a manutenibilidade do código desenvolvido, possibilitando a reutilização de métodos e a implementação de novas funcionalidades, assim como a possibilidade de utilizar objetos *mock* para testes de unidade.

Quanto a questão da manutenibilidade do código, é ressaltado no artigo *Effects of dependency injection on maintainability*, onde os autores Ekaterina Razina e David Janzen (2007) explicitam que a manutenção de código consome por volta de 70% do

---

[15] DAO Pattern - *Data Access Object Pattern*

[16] Personalidade famosa na área de desenvolvimento de software, autor de livros famosos como "Código Limpo", "Arquitetura limpa" e um dos dezessete signatários do "Manifesto Ágil".

[17] Livre tradução do autor

ciclo de vida de um *software* e destacam a eficiência do uso de injeção de dependência como uma importante solução.

Dentro do projeto da *API*, a injeção de dependência é feita da seguinte forma, ao se criar uma classe nova, como um repositório, cria-se um contrato dos métodos que ela deve possuir, ou seja, uma interface (figuras 4 e 5).

```
2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
public class AddressRepository : IAddressRepository
{
    private readonly IDbContextFactory<ApiDbContext> _context;

    0 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public AddressRepository(IDbContextFactory<ApiDbContext> context)
    {
        _context = context;
    }

    2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public List<Address> GetAddresses() => _context.CreateDbContext().Address.Select(x => x).ToList();

    4 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public Address GetAddressById(int id) => _context.CreateDbContext().Address.FirstOrDefault(x => x.Id == id);

    2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public void CreateAddress(Address address)
    {
        var db = _context.CreateDbContext();
        db.Address.Add(address);
        db.SaveChanges();
    }

    2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public void UpdateAddress(Address address)
    {
        var db = _context.CreateDbContext();

        var selectedAddress = db.Address.FirstOrDefault(x => x.Id == address.Id);

        if (selectedAddress != null)
        {
            selectedAddress.Neighbourhood = address.Neighbourhood;
            selectedAddress.Number = address.Number;
            selectedAddress.Street = address.Street;
            selectedAddress.Adjunct = address.Adjunct;
            selectedAddress.CEP = address.CEP;

            db.SaveChanges();
        }
    }

    2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public void DeleteById(int id)
    {
```

Figura 4. Exemplo de classe utilizando uma interface. Fonte: O autor.

```
using Philantropic.EntitiesModels;
using System.Collections.Generic;

namespace Philantropic.Repositories
{
    4 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
    public interface IAddressRepository
    {
        2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
        List<Address> GetAddresses();

        4 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
        Address GetAddressById(int id);

        2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
        void CreateAddress(Address address);

        2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
        void UpdateAddress(Address address);

        2 referências | Felipe Toshio, há 67 dias | 1 autor, 1 alteração
        void DeleteById(int id);
    }
}
```

Figura 5. Exemplo de interface da classe da Figura 4. Fonte: O autor.

Após se cadastrar a injeção de dependência (figura 6), utilizando algum *container*. Neste caso usa-se o nativo do *.net framework* e por último se traz um objeto

do tipo da interface como parâmetro no construtor da classe, que alimentará uma implementação que utilizará os métodos presentes na interface.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IAddressRepository, AddressRepository>();
    services.AddSingleton<IUserMainRepository, UserMainRepository>();
    services.AddSingleton<IDonatorRepository, DonatorRepository>();
    services.AddSingleton<IEntityRepository, EntityRepository>();
    services.AddSingleton<IDonationRepository, DonationRepository>();
    services.AddSingleton<IProjectRepository, ProjectRepository>();
```

Figura 6. Local onde se cadastra todas as dependências do *Dependency Injection Pattern*. Fonte: O autor.

Por último, dentro do projeto da API, foi escolhido seguir uma estrutura de pastas por funcionalidade, não por domínio, devido ao tamanho atual do projeto, sendo feita a divisão por tipo de classe.

2.4. Da infraestrutura

Em nossa pesquisa foi definido pela utilização de um banco de dados relacional, utilizando o *SQL Server* como o *SGBD* [18], como o próprio nome indica, utiliza a linguagem *SQL*. Esta decisão foi tomada pelo fato de ser um projeto embrionário porém que tende a crescer muito, o que futuramente em questões de performance, beneficia um banco relacional. O banco está hospedado dentro da *Azure*, a plataforma nuvem da Microsoft.

A disposição de utilizar a plataforma *Azure* para publicar a *API* se deu devido a sua facilidade de comunicação entre linguagem e infraestrutura. A partir do portal do *Azure DevOps*, é possível realizar os diagnósticos, com relatórios e *logs* do sistema. Utilizando o mesmo *resource group* fica mais simples gerenciar o banco com a *API*. Afinal o plano escolhido para a *API* e o do banco de dados é diferentes, tornando difícil gerenciar os gastos de ambos ao mesmo tempo.

2.5. Do *front end*

O *frameworks* escolhido para o *front-end*, após estudo de mercado, foi o *react-native framework*. O *react-native* é uma combinação dos desenvolvimentos nativos que são feitos para o aplicativos móveis com a famosa biblioteca *Javascript*, o *React*. Desenvolvido pelo *Facebook*, esta biblioteca é uma das mais utilizadas no mercado para criação de aplicativos móveis e possibilita o desenvolvimento para ambos sistemas operacionais, iOS e Android.

A escolha do *framework* foi feita por conta do desempenho que ele apresenta em relação aos seus concorrentes, sendo não tão performático quanto o desenvolvimento nativo de cada sistema, porém como dito pelo autor Rasmus Eskola, em seu artigo *React Native Performance Evaluation* na citação: "*Foi concluído que algumas vezes o React Native pode necessitar de mais recursos do sistema que uma aplicação Android nativa. Porém foi concluído, também, que esta é uma troca muito aceitável considerando a facilidade de desenvolvimento para ambos Android e iOS*

[18] SGBD - Sistema de Gerenciamento de Banco de Dados

usando React Native"[19]- Rasmus Eskola (2018).

Além da arquitetura empregada ao *React Native*, feita por módulos que facilita tanto no desenvolvimento quanto na manutenção do sistema, podendo também reutilizar muitas das funcionalidades e componentes criados.

Outro ponto crucial para a escolha foi devido a utilização de uma ferramenta de depuração *Expo*. A partir dela é possível testar, em tempo real, todas as mudanças feitas no código no celular, sem a necessidade de emuladores ou até mesmo conectar fisicamente o celular ao computador. A conexão com o celular é feita por meio de um *QR code* gerado durante o *build* (Figura 7) que conecta o celular ao servidor local em que a aplicação está rodando.

Figura 7. Exemplo do *build* do aplicativo que gera um *QR code* para se conectar em tempo real pelo celular com a aplicação. Fonte: O autor.

O *Expo*, além de depurar a aplicação, oferece também a possibilidade de *hot reload* que atualiza automaticamente a cada mudança feita no código no celular, facilitando o desenvolvimento do aplicativo.

Após instalar as dependências e testar pela primeira vez o aplicativo, foi necessário realizar sua conexão com a *API* em todas as operações sincronizadamente. Para isso, foi utilizada uma biblioteca *JavaScript* chamada *axios*, e, com ela, é possível se conectar a qualquer API que esteja hospedada na internet, desde que se tenha o acesso da mesma.

O evento teste apresentou problemas na conexão com a *API* que, no momento de execução, se encontrava no mesmo *localhost* que o aplicativo, conferindo a necessidade de abriga-lo na *Azure*.

A abordagem escolhida para utilizar o *axios* foi criar um componente comum que abrigasse essa ligação com *API*. De forma a realizar uma chamada, não necessitando reescrever o *link* (figura 8). Assim, quando é necessário realizar um *request* só há necessidade de chamar esse componente e colocar o *endpoint* desejado como demonstrado na (figura 9).

[19] Livre tradução do autor

```
import axios from 'axios';

export default axios.create({
    baseURL:'https://philantropicapi.azurewebsites.net/api',
    proxy: false
});
```

Figura 8. Código de conexão com a API a partir do aplicativo. Fonte: O autor.

```
import React, {useState} from "react";
import {View, Text, StyleSheet} from 'react-native';
import SearchBar from "../components/SearchBar";
import PhilantropicApi from "../api/PhilantropicApi";

const GetAllEntities = () => {
    const [term, setTerm] = useState('');
    const [results, setResults] = useState([]);

    const searchApi = async () => {
        try{
            const response = await PhilantropicApi.get('/entity');
            console.log(response);
            setResults(response.data)
        }catch(error){
            console.log(error)
        }
    }
```

Figura 9. Código da página de visualizar todas entidades. Fonte: O autor.

2.6. Dos testes

Após o desenvolvimento de cada etapa da aplicação, será necessário um teste, nem que seja manual. Então foram realizados inúmeros testes deste tipo e, por conta de um funcionamento inesperado, optou-se por realizar um teste de carga.

Ao utilizar um teste de unidade, foram realizadas cerca de 150 chamadas simultâneas ao banco de dados no *endpoint* de criação de usuário, o que, na época, permitia por excessão, criação duplicada.

Tal ação ocasionou um problema na infraestrutura do projeto e gerou um mal funcionamento do sistema como um todo, demonstrando a necessidade de escalar o *DevOps* e refatorar algumas partes do código.

3. Funcionamento do Aplicativo Móvel

O aplicativo desenvolvido para o projeto tem a principal funcionalidade de gerenciar doações, como já exposto anteriormente. Inicialmente o usuário ao entrar no aplicativo tem a opção de fazer o *login* no sistema ou se cadastrar, como pode ser observado na (figura 10). Na tela de cadastro, é possível optar por doador ou entidade filantrópica. Caso escolha doador, o primeiro campo indicará o CPF e, no caso da entidade, será o CNPJ.

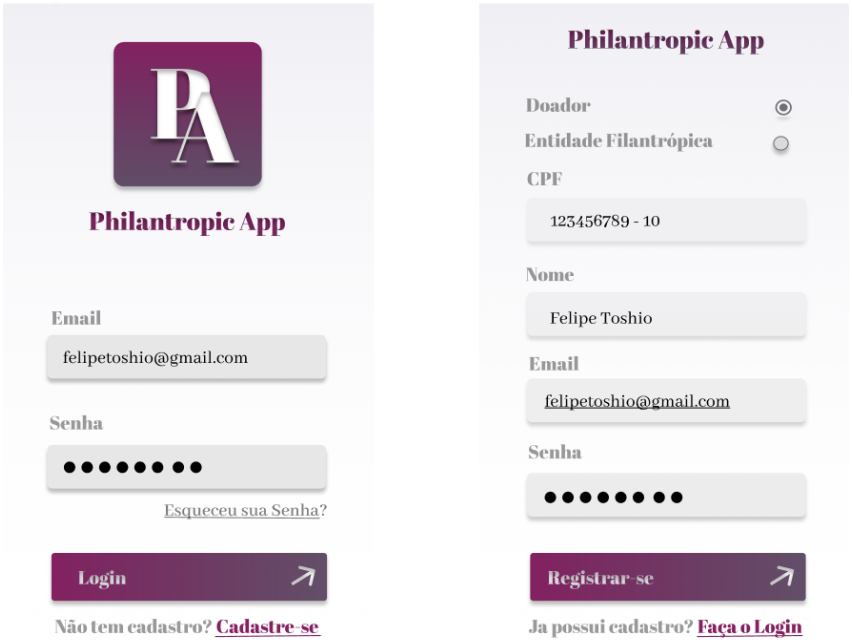

Figura 10. Interface de Login e interface de cadastro no sistema. Fonte: O autor.

No caso de registro, o usuário será levado para a tela de cadastro de localização. Esta podendo ser feita via localização do aparelho ou introdução dos dados manualmente (Figura 11).

Figura 11. Interface de cadastro de endereço. Fonte: O autor.

Após o registro do usuário inicial ou *login*, será apresentado à tela inicial onde será possível visualizar as miniaturas de possibilidades de doações e projetos sazonais das instituições, bem como, a barra de pesquisa para a localização de todas as instituições cadastradas. Em todo momento, enquanto *logado* no aplicativo, é possível

ter acesso às quatro opções de botão que ficam no rodapé do aplicativo. O primeiro volta para a esta tela inicial, o segundo abre as notificações e mensagens, o terceiro abre a tela de "minhas doações" e o último abre a tela de perfil do usuário (Figura 12).

Figura 12. Interface da tela inicial do aplicativo. Fonte: O autor.

Ainda dentro da tela inicial, é possível clicar no nome do instituto e ir para a página da entidade (Figura 13) em que é possível visualizar os projetos e doações em aberto. Ao clicar na última opção do rodapé, ir para a tela de perfil de usuário. Só é possível visualizar o próprio perfil, não sendo permitido navegar pelo de outro usuário doador.

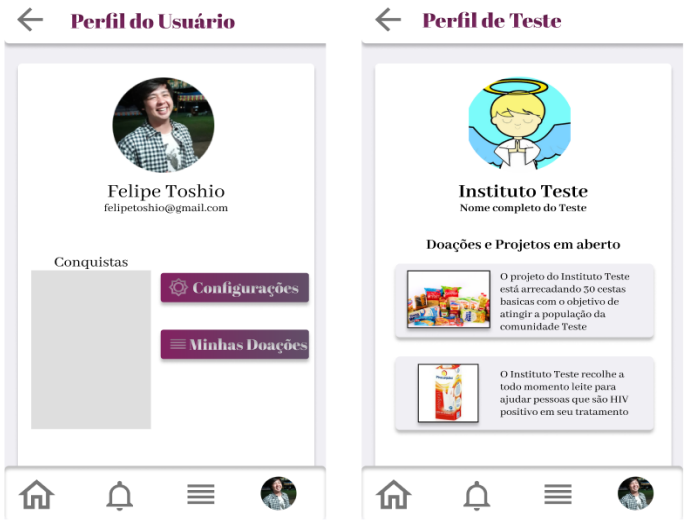

Figura 13. Interfaces de perfil de usuário e de perfil de entidade filantrópica. Fonte: O autor.

Dentro do seu perfil, é possível alterá-lo ao clicar em configurações e entrar na tela de "minhas doações" para verificar suas doações ativas (Figura 14).

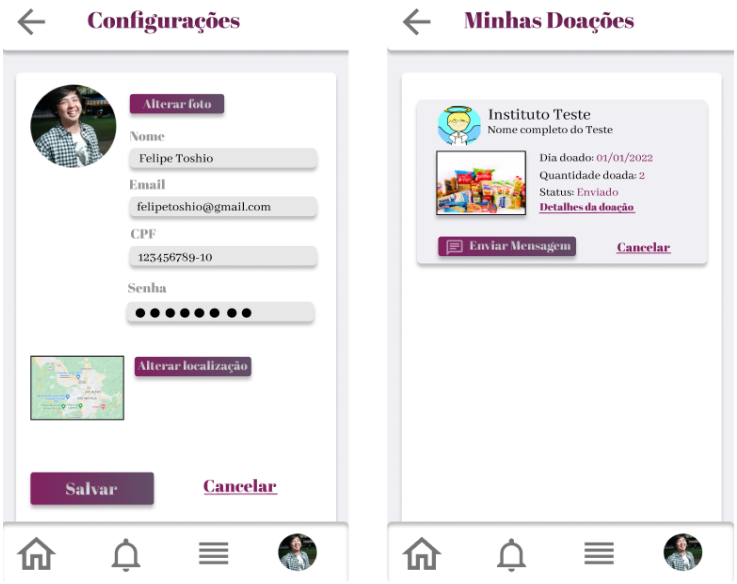

Figura 14. Interfaces de configurações e "minhas doações". Fonte: O autor.

Voltando à tela inicial, ao visualizar uma doação ou projeto, é possível clicar em duas opções "ver mais" onde é exibido detalhes do pedido de doação e "doar agora" onde o usuário é movido para a tela do processo de doação, na qual pode selecionar a quantidade de objetos que quer doar e como será feita a doação, lembrando que somente é possível doar os produtos solicitados pela instituição.

A doação pode ser feita por meio do produto ou pagamento via pix. Este gera um valor de acordo com o preço cadastrado do produto e a quantidade de produtos a ser doado (Figura 15).

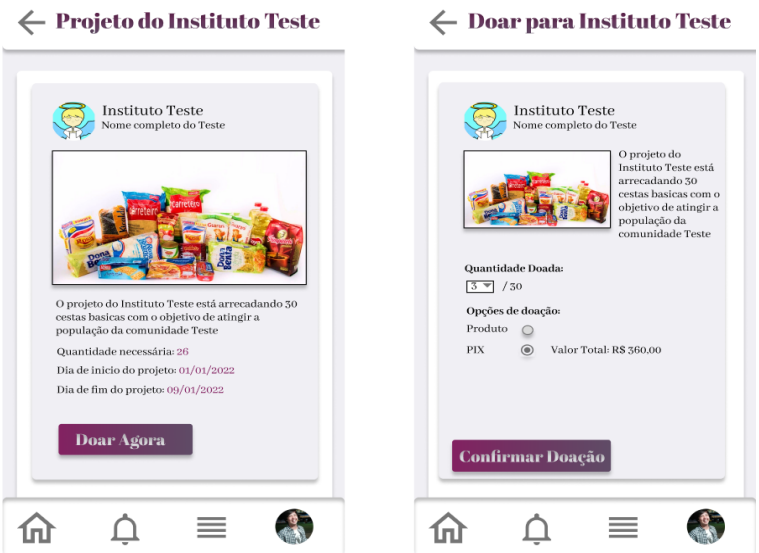

Figura 15. Interface de detalhes da doação e envio da doação. Fonte: O autor.

4. Resultados e discussões

Assim como qualquer projeto de pesquisa, este é um trabalho contínuo que está sujeito a ampliações e reduções em acordo com sua finalidade. Na compreensão de ser um projeto acadêmico, com viabilidade de execução de mercado. Este pesquisador se propõe a implementar o protótipo para avaliação inicial como *MVP*. Sendo assim, na possibilidade de futuras intervenções e de aplicabilidade a partir de sua utilização pelos usuários, possibilitará o desenvolvimento de novas interfaces e perspectivas estruturais deste aplicativo.

E neste ponto, ressaltamos que o aplicativo ainda apresentou alguns problemas de performance em operações custosas feitas ao banco de dados em testes de carga. Entretanto, a aplicação construída possui uma boa experiência em questão de simplicidade de interface e a usabilidade é bem intuitiva. A pesquisa para o uso de cores do *app* levou em consideração o uso de tons de branco como *background* e roxo e sua paleta de cores, com o objetivo de trazer à tona o significado de segurança e honra dada à cor desde os tempos mais antigos, por ser uma cor real.

Entretanto, não foi possível realizar testes com as instituições entrevistadas, pois o aplicativo não atingiu a maturidade técnica necessária para tal. Alguns ajustes ainda são necessários antes de apresentar e receber os devidos *feedbacks*.

Afinal, nem todos os testes foram finalizados. Existem muitos do espectro da pirâmide de testes de *software* que se fazem necessários para continuar, bem como, a presença de outro especialista, para não realizar testes "viciados".

Estes testes ocorrem, por conta da presença de um desenvolvedor, que durante a criação de uma aplicação ou *feature*, acaba seguindo um padrão de testes que não abrange todos os possíveis cenários, evitando pontos críticos de *bug*. Se faz necessária sempre uma "segunda opinião" para qualificar os cenários testados.

Nesta intenção este projeto será apresentado a empresas em busca de voluntários para auxiliar tanto na parte técnica quanto como administradores do sistema.

Além de que a escolha foi uma evolução desde a primeira interface que de acordo com usuários era confusa para pessoas mais velhas e daltônicas (Figura 16).

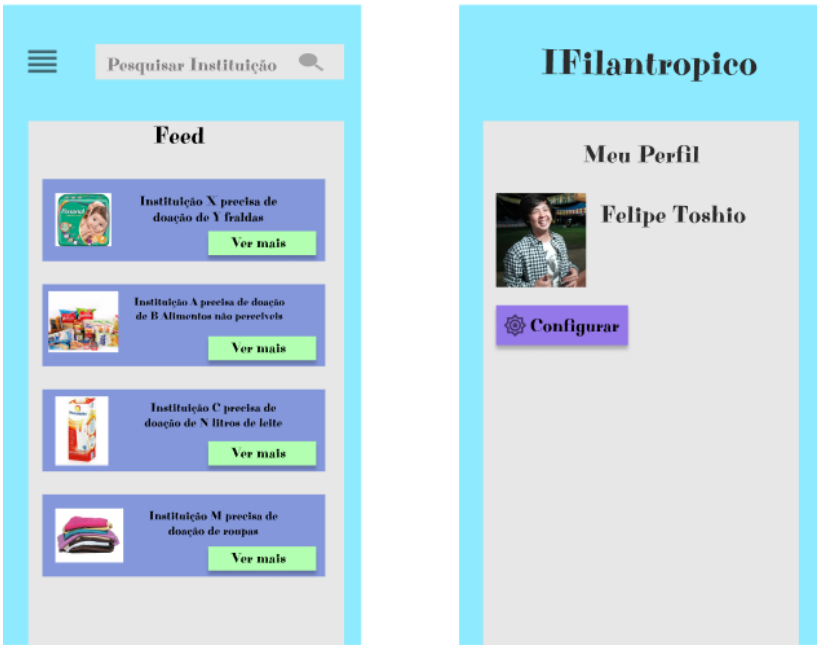

Figura 16. Primeira interface criada para a aplicação *mobile*. Fonte: O autor.

A interface inicialmente é focada para o público de dispositivos com sistema operacional Android.

5. Trabalhos futuros

No contínuo desenvolvimento de novas versões, serão realizadas melhorias no âmbito de infraestrutura e design, bem como, de versões atualizadas das ferramentas já apresentadas anteriormente, visando a preservação da qualidade do aplicativo.

Conforme o projeto inicial, se propõe a realizar mais testes aprofundados focando os *corner cases*, que são casos fora do fluxo normal de utilização do aplicativo, a fim de encontrar possíveis *bugs* antes que o usuário final use o sistema como um todo. Juntamente com a necessidade de aumentar o número de testes de unidade, implementar testes de interface e integração para garantir uma boa qualidade de aplicativo.

Aproveitando que a ferramenta de desenvolvimento utilizada permite o desenvolvimento híbrido, possibilita a integração do aplicativo para dispositivos iOS.

Por fim, como já explicitado acima, melhorar o MVP e focar também nas demandas de *stretch* definidas.

6. Conclusões

Como projeto de pesquisa, não poderia deixar de ressaltar a importância deste para este pesquisador que vislumbrou sua possibilidade ainda nos primeiros períodos do Bacharelado de Sistemas de Informação, o quão crucial foi criar um documento de requisitos e estudar bem o que seria feito no aplicativo antes de iniciá-lo.

Explicitar exatamente o que seria feito antes de realmente começar a desenvolver, promoveu discussões sobre o seu desenvolvimento e facilitou o entendimento que se tem sobre o produto em questão. Entretanto este trabalho é um passo inicial e importante de um projeto que pode vir a ser uma importante ferramenta

para aqueles que se dedicam às questões sociais, intenção que move a muitos diante de um modelo socioeconômico que nos difere e nos isola em camadas distintas.

Ao desenvolver esta ideia, não temos a pretensão de solucionar as questões acima expostas, mas sim de buscar junto à tecnologia, democratizar e defender uma sociedade mais justa e igualitária, papel de quem teve a oportunidade de pesquisar em uma faculdade pública e de qualidade. Assim sendo, aplicações como esta, sempre terão espaço para melhorar a qualidade de vida da sociedade como um todo, observando uma melhor integração entre doador e receptor. Entidades como as pesquisadas, precisam de alternativas para melhor atender seu público-alvo, dando a eles a oportunidade de se sentirem cidadãos.

Referências

ABAN: Associação dos Amigos. Disponível em: . Acesso em 21 de janeiro de 2022.

Learn about .NET Framework, a development platform for building apps for web, Windows, and Microsoft Azure. Disponível em: <https://docs.microsoft.com/en-us/dotnet/framework/> . Acesso em 26 de janeiro de 2022.

Introduction React Native: Getting Started. Disponível em: <https://reactnative.dev/docs/getting-started>. Acesso em 26 de janeiro de 2022.

Figma: Guide to Developer Handoff. Disponível em: <https://www.figma.com/best-practices/guide-to-developer-handoff/>. Acesso em 26 de janeiro de 2022.

About Api Management Concepts. Disponível em: <https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts> Acesso em 26 de janeiro de 2022.

OAuth 2.0 . Disponível em <https://oauth.net/2/>. Acesso em 31 de janeiro de 2022.

Halili, Festim, and Erenis Ramadani. "Web services: a comparison of soap and rest services." Modern Applied Science 12.3 (2018): 175.

Albahari, Ben, Peter Drayton, and Brad Merrill. C# Essentials: Programming the. NET Framework. " O'Reilly Media, Inc.", 2002.

Marinescu, Floyd. EJB design patterns. New York: Wiley, 2002.

Chochlık, Matúš. "Implementing the factory pattern with the help of reflection." Computing and Informatics 35.3 (2016): 653-686.

Razina, Ekaterina, and David S. Janzen. "Effects of dependency injection on maintainability." Proceedings of the 11th IASTED International Conference on Software Engineering and Applications: Cambridge, MA. 2007.

Eskola, Rasmus. "React Native Performance Evaluation." (2018).

GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos

Martin, Robert C., and Micah Martin. *Agile principles, patterns, and practices in C# (Robert C. Martin)*. Prentice Hall PTR, 2006.

Lohmann, Roger A. "Charity, philanthropy, public service, or enterprise: what are the big questions of nonprofit management today?." *Public Administration Review* 67.3 (2007): 437-444.