DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de plataforma online de doações baseada em uma arquitetura DDD, Event Sourcing e CQRS

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Leonardo Schevz de Werk

Florianópolis, Dezembro de 2017

Desenvolvimento de plataforma online de doações baseada em uma arquitetura DDD, Event Sourcing e CQRS

Leonardo Schevz de Werk

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Rodrigo Castelan Carlson

Banca Examinadora:

Lucas Pelegrino
Orientador na Empresa

Rodrigo Castelan Carson
Orientador no Curso

Prof. Ricardo José Rabelo
Responsável pela disciplina

Cleber Jorge Amaral, Avaliador

Pablo Pompillio Andreus, Debatedor

Henrique Eduardo Felipini, Debatedor

Agradecimentos

Agradeço a realização deste trabalho, assim como do longo ciclo de minha vida que se encerra com ele, primeiramente a todos os professores que participaram do processo de minha graduação, particularmente àqueles que exerceram um papel essencial de apoio e amizade. Agradeço especialmente ao Professor Rodrigo Carlson, pela orientação neste trabalho.

Aos meus colegas da Doare, pelo aprendizado ímpar adquirido no curto período de contato que tivemos na Doare. Sou grato ao Lucas Pelegrino, por toda a dedicação e paciência em me treinar para colaborar com o grupo.

Agradeço imensamente aos meus pais, Miriam e Antonio, meu irmão André, minha tia Liane, e toda minha família, por serem o porto seguro sem o qual eu não teria chego até aqui.

Agradeço também aos meus caros amigos Lucas, Wagner, Conrado, Victor, Gabriel, Astor, Fernando, Rodrigo e Diego, por serem a minha segunda família, que esteve sempre ao meu lado nas horas felizes e difíceis.

Por último, mas não menos importante, obrigado aos meus queridos colegas do curso, Renê, Pablo, Michael, e tantos outros, pela parceria no enfrentamento dos desafios desses anos, e também por torna-los incríveis.

Resumo

O terceiro setor, com suas organizações não governamentais (ONGs) e demais instituições não lucrativas, é responsável por tentar suprir necessidades da sociedade que não são supridas pelos governos nem pela iniciativa privada. O campo de atuação desse tipo de instituição vai desde de cuidados com crianças em situação de risco em uma comunidade até ações mundiais de proteção ao meio-ambiente. Um aspecto, no entanto, que une muitas dessas instituições são os esforços de arrecadação de doações para garantir sua existência.

Em busca de tornar menos oneroso o processo de arrecadação de doações, muitas instituições buscam soluções através das quais seus apoiadores possam realizar suas contribuições de maneira prática do conforto de suas casas. Algumas soluções presentes na internet, comumente chamadas de *crowdfunding*, ou financiamento coletivo, serviram como ferramentas iniciais às quais essas instituições poderiam recorrer, mas eram soluções genéricas, e o mercado carecia de uma solução especializada na arrecadação de doações para esse tipo de instituição.

Nesse contexto surgiu em 2012 a Doare. Com sua plataforma de doações online, a empresa criou um portal no qual ONGs podem expor informações sobre seu trabalho e receber doações pela internet. Nesse portal, um doador que gostaria de doar para uma ONG mas não sabe qual pode procurar uma cuja atuação ele simpatize, e realizar uma doação.

Este trabalho é o resultado e registro de um estágio realizado na empresa. Na duração desse estágio ocorreu a participação no desenvolvimento e manutenção da plataforma de doações, de outros produtos que a empresa criou desde sua fundação e da infraestrutura de *software* na qual tudo isso se baseia. Essa colaboração proporcionou o contato com diversas tecnologias e metodologias de desenvolvimento de aplicações para a internet, entre as quais se destaca a arquitetura aplicada pela empresa, baseada nos modernos conceitos *Domain Driven Design*, *Event Sourcing* e CQRS.

Palavras-chave: *Domain Driven Design*, *Event Sourcing* CQRS, Doações Online, Desenvolvimento Web, *Front-end*, *Back-end*.

Abstract

The third sector, with its nongovernmental organizations (NGOs) and other non-profit institutions, is responsible for trying to supply some of society's needs that are not supplied by the governments nor by the private initiative. The field of activities of this kind of institutions ranges from taking care of children at risk to actions to protect the world's environment. One aspect, nevertheless, that many of these institutions have in common is the effort to collect donations that make their existence possible.

In an attempt to make the process of collecting donations less costly, many institutions look for solutions that make it possible for their supporters to contribute in a practical way, from the comfort of their homes. Some solutions already available on the internet, usually called crowdfunding platforms, served as starting tools that could be used by these institutions, but they were generic fundraising tools, and the market was in need of a solution that was specialized in the collection of donations to this kind of institution.

In this context, in 2012, Doare was created. With its online donations platform, the company created a portal where NGOs could display information about their works and receive donations via the internet. In this portal, a donor that would like to donate to an NGO but doesn't know which could look for one whose activities are of his or her interest, and donate.

The present work is the result and report of an internship that took place in the company. In the duration of this internship, there was participation in the development and maintenance of the donation platform, of other products created by the company since it's founding, and of the whole software infrastructure on which they are all based. This collaboration has provided the contact with several technologies and methodologies of web development, among which the architecture implemented by the company, based on modern concepts such as Domain Driven Design, Event Sourcing and CQRS, stands out.

Keywords: *Domain Driven Design*, *Event Sourcing* CQRS, Doações Online, Desenvolvimento Web, *Front-end*, *Back-end*.

Lista de ilustrações

Sumário

1 Introdução

1.1 Apresentação do Problema

Organizações não governamentais (ONGs) que buscam prestar serviços em diversas áreas para a comunidade de maneira não lucrativa dependem fortemente, e muitas vezes exclusivamente, de doações realizadas por pessoas que apoiam a causa e o trabalho dessas organizações. Para que esse suporte financeiro alcance as instituições beneficiadas, muito trabalho é necessário, desde cuidados com a imagem pública da instituição e campanhas para arrecadação de verba, até a organização do recebimento de um volume muito grande de doações. A maioria dessas doações são de valor relativamente baixo, e são muito frequentemente realizadas de maneira presencial, com funcionários de ONGs precisando ir até a casa dos doadores para recolher as contribuições e deixar recibos.

Todo esse processo gera um custo muito grande para as campanhas, e acaba por consumir uma parte valiosa dos recursos arrecadados. Em busca de resolver tal problema e encontrar uma solução mais eficiente de execução de campanhas de doações, algumas instituições buscam na informática uma maneira de tornar o processo menos custoso, de forma a criar páginas online de doação onde aqueles que desejam contribuir podem, do conforto de suas casas, realizar doações utilizando um cartão de crédito ou alguma outra forma de pagamento.

Com a possibilidade de ter seus apoiadores realizando doações online, muito do custo de recursos utilizados nesse processo pode ser cortado. No entanto, somente ONGs de porte relativamente grande conseguem implementar tal serviço, uma vez que desenvolver sistemas de pagamento online não é uma tarefa simples, e esse com certeza não é o foco dessas instituições, que em sua maioria nem possuem uma equipe técnica capacitada para a tarefa.[1]

1.2 Justificativa

Enquanto a criação de sistemas de doações online é uma tarefa relativamente complicada para ser realizadas de maneira individual por uma ONG, tal sistema é uma necessidade compartilhada de muitas dessas instituições, e o problema que cada uma delas precisa resolver é semelhante. Isso justifica a criação de soluções mais genéricas, que

[1] O conhecimento sobre as dificuldades de ONGs em arrecadar doações antes da existência de meios *online* de arredação foi obtido através de conversas com os membros da empresa, que realizaram pesquisas de mercado durante sua fundação.

atendam às necessidades dessas instituições da melhor maneira possível, e que possam ser adotadas por elas sem que precisem despender grandes recursos financeiros e de tempo.

A demanda por uma ferramenta centralizada de doações para organizações não governamentais é o que justifica o projeto que deu origem ao presente trabalho. A empresa Doare, na qual está sendo realizado o estágio que possibilitou a participação no projeto, busca criar soluções para que as ONGs possam receber doações.

Da perspectiva da engenharia de controle e automação, o desenvolvimento dessas soluções oferece um ambiente riquíssimo onde diversos conceitos podem ser utilizados e aprimorados, particularmente na área de informática. Conhecimentos de modelagem e arquitetura de software, bancos de dados, programação assíncrona, entre outros, precisam ser revisados, aprofundados e utilizados no desenvolvimento da plataforma.

1.3 Objetivos

A plataforma de doações da Doare já é utilizada por um número considerável de ONGs, que a utilizam como principal meio para obter doações. O objetivo do presente trabalho é dar inicio a uma documentação detalhada, ainda não existente, de todo o sistema com o qual a empresta trabalha, assim como à modelagem do sistema e das novas funções implementadas. No futuro, com essa documentação em mãos, será possível utiliza-la como referência para compreensão do sistema.

Junto a essa documentação, objetiva-se contribuir para o aprimoramento e manutenção contínuos do serviço, por meio da revisão e atualização do software existente, assim como a implementação de novas funcionalidades para que seja possível atender às necessidades das instituições de maneira ainda mais completa.

1.4 Metodologia

Embora o software já esteja em uso, ainda não foi formalmente criada a especificação do projeto. Visa-se, portanto, com esse trabalho, primeiramente especificar as necessidades que devem ser atendidas pela plataforma, de forma a entender aspectos que podem ainda estar faltantes, assim como reconhecer as prioridades de desenvolvimento adicional para o escopo do projeto.

A metodologia utilizada para a especificação e modelagem do sistema foi a Linguagem de Modelagem Unificada UML, que fornece uma série de padrões simbólicos para que engenheiros de software definam as características de um sistema. Foram utilizados alguns diagramas dessa linguagem para detalhar casos de uso da aplicação, assim como especificações do projeto. [2]

Para planejamento de tarefas de desenvolvimento foi aplicada a metodologia de desenvolvimento ágil SCRUM, na qual o time de desenvolvimento define para um período de tempo tarefas que devem ser realizadas. Ao fim desse período, que dura tipicamente de uma semana até um mês, os membros do time se reunem para avaliar a produtividade de cada um, entender quais metas foram atingidas, e em seguida determinar as tarefas para o próximo ciclo. [3]

1.5 Estrutura do documento

O presente documento está estruturado da seguinte maneira: no Capitulo 2 são discutidas as especificações de requisitos do projeto, tanto para a interface gráfica dos produtos oferecidos pela empresa quanto para a API que é utilizada por elas. Nos capítulos seguintes será descrito o modelo da solução implementada. Será descrita a aplicação do usuário, desenvolvida em Angular 4, no Capitulo 3, seguida pela descrição da API (*Aplication Programming Interface*, ou Interface de Programação de Aplicações) da Doare, no Capitulo 4. No Capitulo 5 são descritas em maior detalhe as atividades realizadas durante este trabalho, e no Capitulo 6 conclui-se o documento com considerações finais.

2 Levantamento e Análise de Requisitos

A Doare oferece desde 2012 diferentes soluções de software relacionadas ao auxílio na captação de recursos para Organizações não governamentais. Esses serviços - diversas aplicações para a web com diferentes interfaces gráficas - utilizam uma API que já está em funcionamento. Essa API é mantida pela equipe de desenvolvimento da empresa e compõe, junto à infraestrutura de software sob a qual é baseada, o sistema de processamento de doações da Doare. Esse sistema é composto, por sua vez, de dois subsistemas: Um sistema legado, desenvolvido em PHP, que ainda é responsável principalmente pela estrutura da aplicação que tange ao painel ao qual o representante de uma instituição tem acesso para acompanhar o andamento de sua captação - paralelamente ao sistema legado, um novo sistema está sendo desenvolvido com o uso da tecnologia *node.js*, um interpretador de *javascript* que vem sendo largamente utilizado para desenvolvimento de aplicações para a web, tanto para o *front-end*, com *frameworks* como o *angular* 4, quanto para o *back-end*, na criação de APIs e microserviços que funcionam como o cérebro das aplicações. O fato de o sistema legado ainda estar em uso faz com que uma parte dos trabalhos de desenvolvimento realizados tenha sido voltado para a integração do sistema novo com o sistema antigo, assim como a transferência e implementação de algumas funcionalidades do sistema antigo no sistema novo, já que deseja-se no futuro ter toda o sistema implementado em *node.js*. Ambos os sistemas foram desenvolvidos sem que antes tivessem sido especificados e analisados formalmente os requisitos.

Tratando-se de uma empresa pequena, o software sempre foi desenvolvido utilizando metodologias ágeis, de forma a atender as demandas que surgiam conforme a aplicação era utilizada e o produto da empresa se adaptava ao mercado. Buscou-se, portanto, nesse capítulo, compreender e formalizar, a partir do funcionamento atual da aplicação e das mudanças que deveriam ser implementadas, a especificação de requisitos do sistema.

Este capítulo foi divido em duas sessões. Na primeira são discutidos os requerimentos das interfaces do usuário e o modo de uso dos produtos da empresa que serão descritos na primeira sessão. Mais do que simplesmente detalhar como é a parte gráfica da aplicação, essa sessão determina como o software deve funcionar da perspectiva do usuário, sendo portanto detalhados os atores do sistema e seus casos de uso. Na sessão 2.2 são especificados os requerimentos da API que serve como estrutura para todos esses serviços. Serão também detalhadas as funcionalidades da API que foram e virão a ser desenvolvidas.

2.1 Especificação de requisitos da Interface do Usuário

A Doare oferece atualmente três serviços que têm como temática central as doações online para ONGs. O mais antigo dele é a página homônima da empresa, pela qual ONGs podem cadastrar suas informações e criar campanhas para as quais pessoas podem realizar doações. Essa plataforma já conta com mais de 2500 ONGs cadastradas e já arrecadou no total mais de R$ 4 milhões em doações. Outro serviço oferecido pela empresa é o Memio. Trata-se de uma plataforma parecida com a Doare, mas no caso do Memio quando um usuário efetua uma doação ele passa a concorrer a participar de uma experiência. Essa experiência pode ser de vários tipos, desde encontrar uma celebridade até participar de um jantar de negócios. O Memio funciona, portanto, como uma plataforma onde ONGs podem criar rifas para arrecadar doações. Por ultimo, a doare oferece o serviço Giveom, que permite que ONGs coloquem uma pequena extensão em suas próprias páginas na internet através do qual elas podem arrecadar doações passando pelo sistema da Doare.

Ainda que os três serviços apareçam para os usuários como ferramentas individuais, na verdade todos eles estão baseados no mesmo sistema de pagamento, e todos eles usam, finalmente, a interface de doações, chamada caixa de doações, como meio para interagir com os usuários no momento do pagamento. De um ponto de vista tecnológico, é como se a página da Doare e do Memio fossem usuários do serviço Giveom, implementando a caixinha de doações em suas páginas.

Para todas as instituições atendidas, ainda existe um painel onde o administrador de uma ONG pode acompanhar o andamento de suas campanhas, verificar quem foram os doadores e requisitar a transferência do valor arrecadado. Toda essa parte do serviço ainda está implementado no sistema legado da empresa.

Neste trabalho foi descrito com detalhes a especificação da caixa de doações, que é o produto principal oferecido e desenvolvido pela empresa atualmente. A maior parte dos esforços de desenvolvimento realizados no escopo deste trabalho foram voltados para a manutenção e melhoria dessa parte do sistema da empresa. Para essa parte da aplicação, foram levantados diagramas de casos de uso, diagramas de sequência e de atividade, assim como a descrição textual detalhada dos casos de uso e atores do sistema.

Outras partes do sistema tiveram seu funcionamento descrito de maneira mais simples, de forma a expor o contexto geral no qual a caixa de doações e a API do sistema estão relacionadas com outras partes da aplicação. São essas: A plataforma da Doare, onde ONGs podem ter um perfil público e arrecadar campanhas, o Memio, onde podem ser realizadas campanhas de experiência e, por fim, o painel do administrador da ONG, onde o usuário de qualquer um desses serviços pode acompanhar suas doações.

2.1.1 A caixinha de Doações - O terminal de pagamentos

O terminal de pagamentos utilizado pelo sistema é uma aplicação relativamente simples do ponto de vista do usuário. Trata-se de uma única página para a qual o doador é direcionado, de onde quer que ele esteja, quando deseja concretizar uma doação. No entanto, esse terminal de pagamentos é multimeio - aceita cartão de crédito, boleto bancário e *paypal*, multi-moeda - aceita até o momento Reais, Dólares e Euros, e multi-idioma, o que aumenta um pouco mais o grau de sofisticação do sistema. Nessa subseção são formalizados os casos de uso dessa parte da aplicação.

Resumidamente, o usuário escolhe um método de pagamento, um idioma e uma moeda, insere seus dados, e aguarda até que a operação seja aprovada ou não. Em todas as modalidades de pagamento, o sistema da Doare se comunica com um terceiro elemento, externo à empresa, que é responsável por administrar e validar as transações. Esses sistemas são comumente chamados de *gateways* de pagamento, e no caso da Doare são utilizados serviços diferentes para diferentes meios de pagamento, como o *Stripe* para pagamentos com cartão de crédito e o moip no caso dos pagamentos com boleto bancário. Nesta sessão serão descritos os casos de uso do ator visitante, que é o único ator desta parte da plataforma.

Um visitante das páginas das ONGs atendidas pela Doare chega no terminal de pagamento sem precisar se autenticar. Na realidade, um sistema de autenticação e um painel personalizado para cada usuário é um funcionalidade ainda não implementada. O doador entra na página de doações, portanto, como visitante. Ao colocar seus dados pessoais e efetuar uma doação, se suas informações ainda não estiverem no sistema, ele se torna um usuário registrado no sistema. A partir daí todas as doações que ele realizar passam a ser vinculadas a ele, ainda que ele não seja diretamente informado disso nem tenha acesso às suas informações de seu registro. Essa informação é útil no painel dos administradores das ONGs, que podem assim acompanhar o apoio de cada pessoa em suas campanhas. Assim sendo, um doador pode realizar doações utilizando um dos diferentes métodos de pagamento possíveis. Ao informar seus dados e os dados de pagamento, o sistema processa o pagamento da maneira adequada para cada forma disponível para pagamento. Na figura 1 estão ilustrados os casos de uso desse ator.

Caso de uso UC01: Realizar doação com cartão de crédito

Um possível doador chega através de diferentes páginas de campanhas na caixa de doações, e define um método de pagamento utilizando uma das abas disponíveis, como pode ser visto na Figura 2 . Nesse caso de uso é escolhido o pagamento com cartão de crédito.

Fluxo primário de eventos:

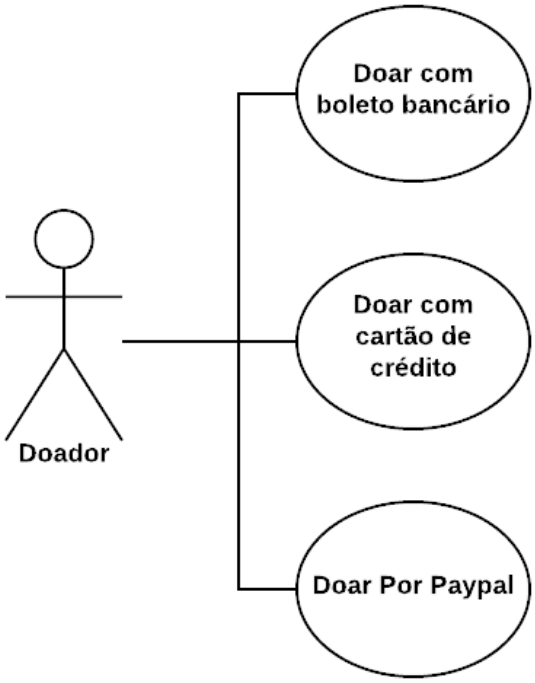

Figura 1 – Casos de uso do Ator Visitante/Doador

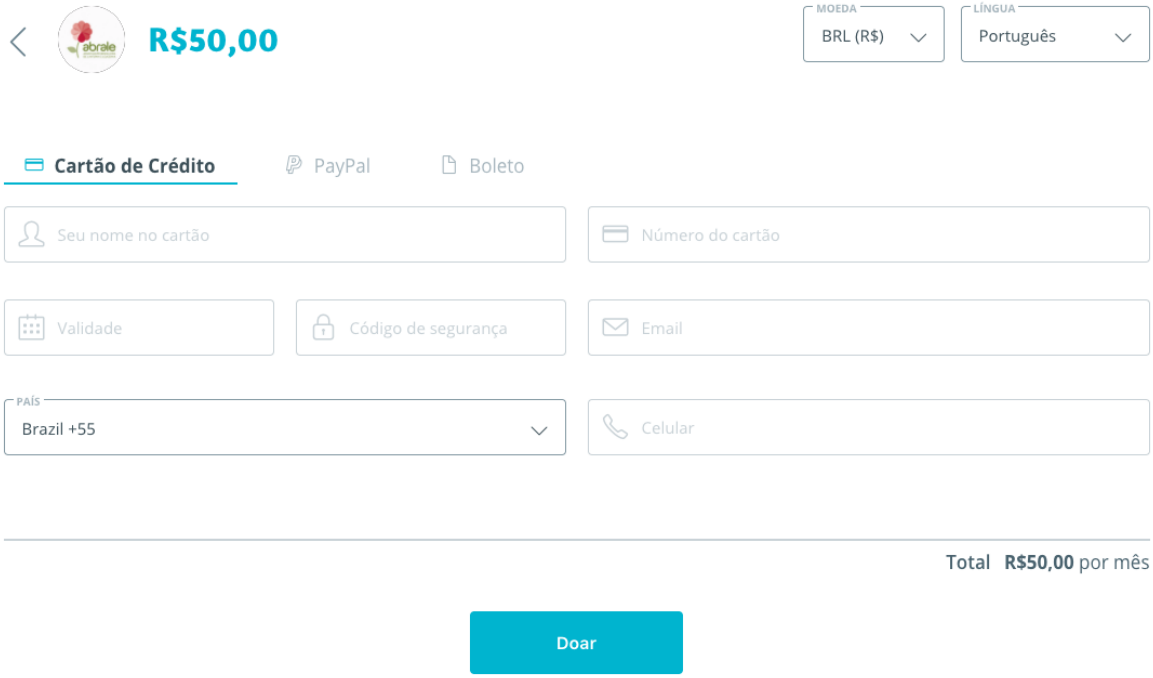

Figura 2 – Caixinha de doações na opção cartão de crédito

1. O doador preenche as informações necessárias para pagamento com cartão de crédito (nome, numero do cartão, email, etc).

2. O sistema processa o pagamento, verificando se os dados são consistentes, e envia os dados de pagamento para o serviço que processa o pagamento com cartão de crédito (*gateway*).

3. O *gateway* autoriza a transação e informa o sistema

4. O sistema recebe a confirmação do pagamento e informa o doador que sua doação foi aprovada. Além disso, um email é enviado para o doador e para o administrador da ONG com um recibo de pagamento.

Fluxo alternativo A: No segundo passo o sistema verifica que os dados do cartão ou os dados pessoais não são consistentes.

1. O sistema informa o doador através da interface gráfica quais campos do formulário foram preenchidos incorretamente.

2. O doador volta ao passo 1 e preenche o formulário novamente.

Fluxo alternativo B: No terceiro passo o pagamento não é autorizado pelo *gateway* de pagamento.

1. O sistema informa o doador através da interface gráfica que o seu pagamento não foi autorizado, e sugere que ele verifique os dados, tente utilizar outro cartão, ou escolher outra forma de pagamento.

2. O doador volta ao passo 1 e preenche o formulário novamente.

Nas Figuras 3 e 4 estão representados, respectivamente, o diagrama de sequência e o diagrama de atividade desse caso de uso.

Caso de uso UC02: Realizar doação com boleto bancário

Neste segundo caso de uso, o início do procedimento é semelhante. O doador preenche dados em um formulário e envia para o sistema. O sistema valida os dados e envia para um *gateway* de pagamento, que pode ser diferente do *gateway* do pagamento com cartão de crédito. Desta vez, no entanto, em vez de responder o sistema com uma confirmação de pagamento, o *gateway* fornece um *link* para a impressão de um boleto bancário. Quando o doador efetua o pagamento, em um outro momento, o *gateway* notifica o sistema com uma requisição *http*, como será detalhado mais a frente nas sessões que tratam a API do sistema. O sistema notificado então finaliza o processamento do pagamento, informando o usuário que a doação foi bem sucedida.

Fluxo primário de eventos:

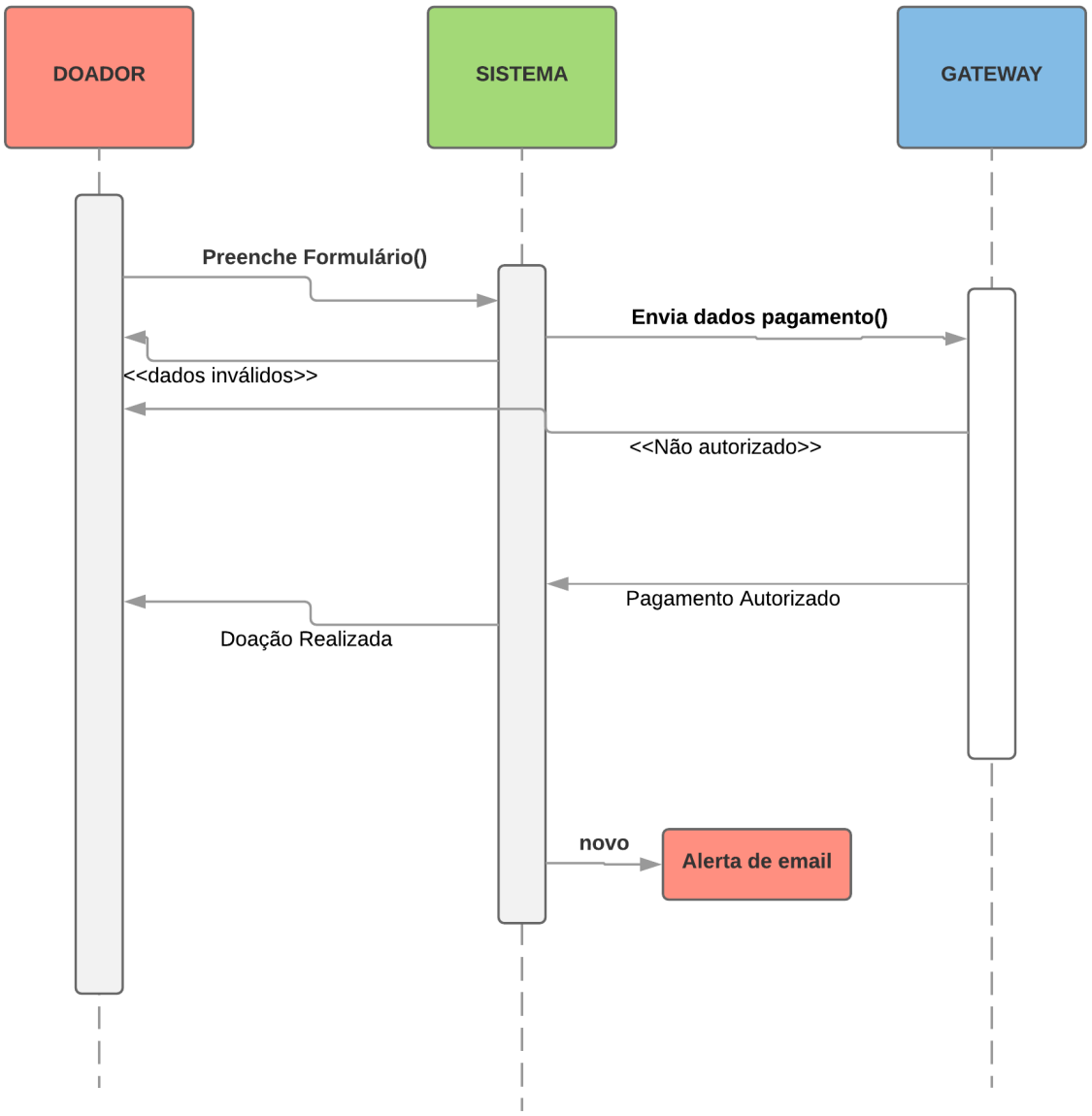

Figura 3 – Diagrama de sequência UC001

1. O doador preenche as informações necessárias para pagamento com boleto bancário (nome, email, etc).

2. O sistema processa o pagamento, verificando se os dados são consistentes, e envia os dados de pagamento para o serviço que processa o pagamento com boleto bancário (*gateway*).

3. O gateway responde com um link para impressão do boleto bancário.

4. O sistema redireciona o doador para a impressão do boleto.

5. O doador paga o boleto com o valor previamente definido.

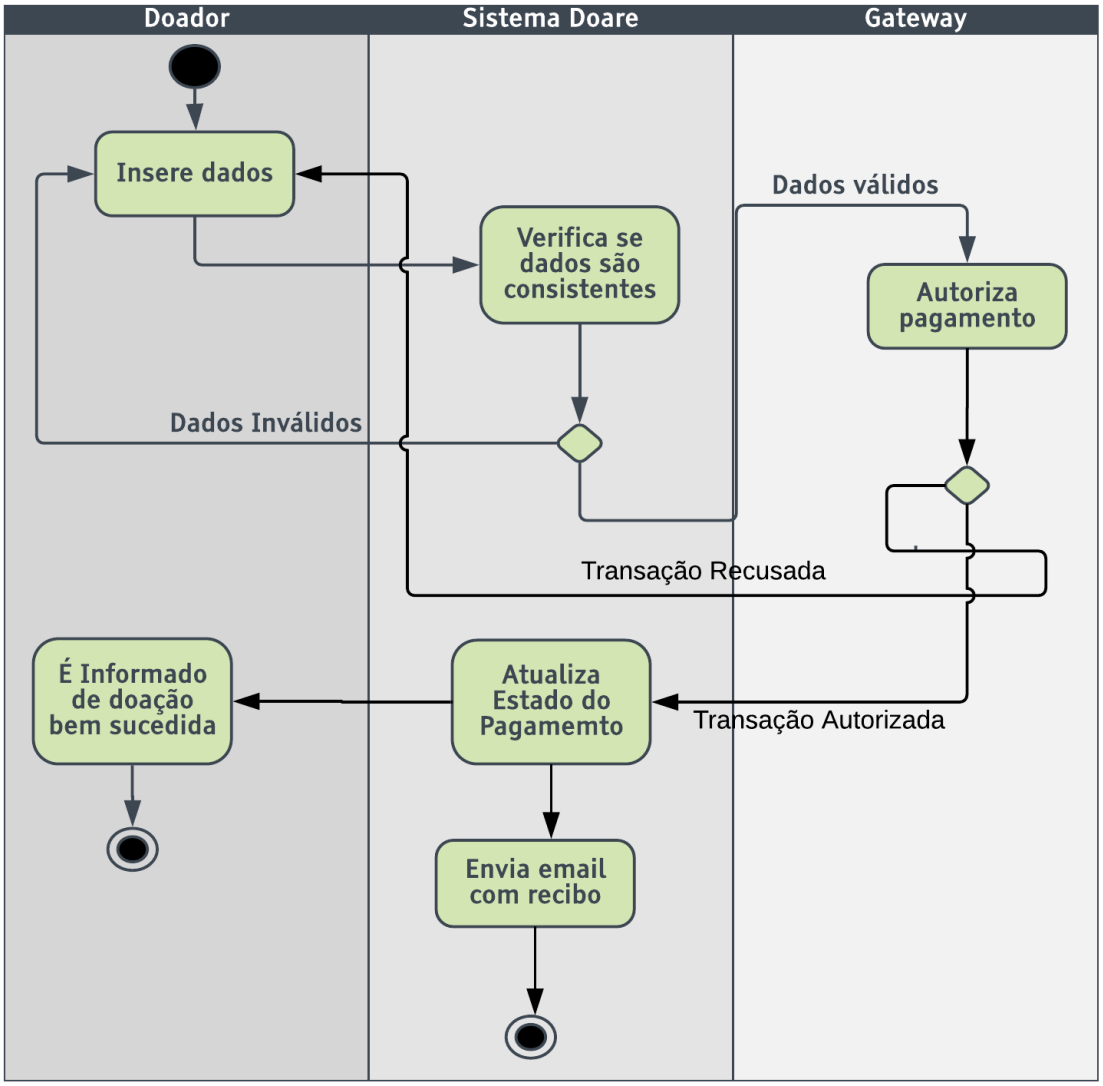

Figura 4 – Diagrama de atividade UC001

6. O *gateway* notifica o sistema que o pagamento foi aprovado.

7. O sistema recebe a confirmação do pagamento e informa o doador que sua doação foi aprovada. Além disso, um email é enviado para o doador e para o administrador da ONG com um recibo de pagamento.

Fluxo alternativo A: No segundo passo o sistema verifica que os dados do cartão ou os dados pessoais não são consistentes.

1. O sistema informa o doador através da interface gráfica quais campos do formulário foram preenchidos incorretamente.

2. O doador volta ao passo 1 e preenche o formulário novamente.

Fluxo alternativo B: No quinto passo o pagamento não é realizado pelo doador até o vencimento do boleto.

1. O *gateway* notifica a falha no processo ao sistema.
2. O sistema informa o doador que o pagamento não foi realizado no prazo, e informa que o usuário deve reiniciar o processo de doação caso queira realiza-la de fato.

Fluxo alternativo C: No quinto passo o pagamento é realizado com um valor acima do valor previamente definido.

1. O *gateway* notifica a API de que o pagamento foi realizado, inclusive com o valor que foi pago.
2. A plataforma informa o doador que o pagamento realizado com sucesso, informando o usuário que o valor pago é maior do que o valor do boleto, e que a quantia paga será direcionada para a campanha da ONG.

Nas Figuras 5 e 6 estão representados, respectivamente, o diagrama de sequência e o diagrama de atividade desse caso de uso.

Caso de uso UC03: Realizar doação com Paypal

O *Paypal* é um serviço online de pagamentos que funciona como uma espécie de carteira digital onde seus usuários podem deixar algum dinheiro armazenado com o qual eles podem realizar com mais praticidades pagamentos online. O processamento de doações com esse serviço ocorre de maneira semelhante ao pagamento com boleto bancário. Após preenchidos os dados de pagamento, o doador é redirecionado o sistema do *paypal* para realizar o pagamento. Uma vez completo o pagamento, o sistema é notificado para dar continuidade no processo, como ocorre com as outras formas de pagamento.

Fluxo primário de eventos:

1. O doador preenche as informações necessárias para pagamento com *Paypal* (nome, email, etc).
2. O sistema processa o pagamento, verificando se os dados são consistentes.
3. O sistema envia os dados para o *Paypal* e redireciona o doador para o sistema do *gateway*.
4. O doador realiza o pagamento no sistema do *Paypal*.
5. O *gateway* notifica o sistema que o pagamento foi aprovado.

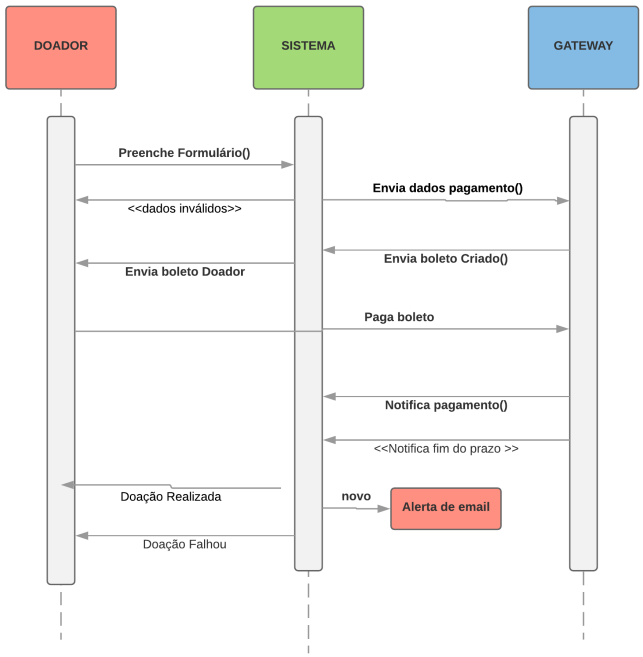

Figura 5 – Diagrama de sequência UC002

6. O sistema recebe a confirmação do pagamento e informa o doador que sua doação foi aprovada. Além disso, um email é enviado para o doador e para o administrador da ONG com um recibo de pagamento.

Fluxo alternativo A: No segundo passo o sistema verifica que os dados do cartão ou os dados pessoais não são consistentes.

1. A plataforma informa o doador através da interface gráfica quais campos do formulário foram preenchidos incorretamente.
2. O doador volta ao passo 1 e preenche o formulário novamente.

Fluxo alternativo B: No quarto passo o pagamento não é realizado com sucesso pelo doador na plataforma do *paypal*.

1. O *gateway (paypal)* notifica a falha no processo à API.
2. O sistema informa o doador que o pagamento não foi realizado com sucesso, e informa que o usuário deve reiniciar o processo de doação caso queira realiza-la de fato.

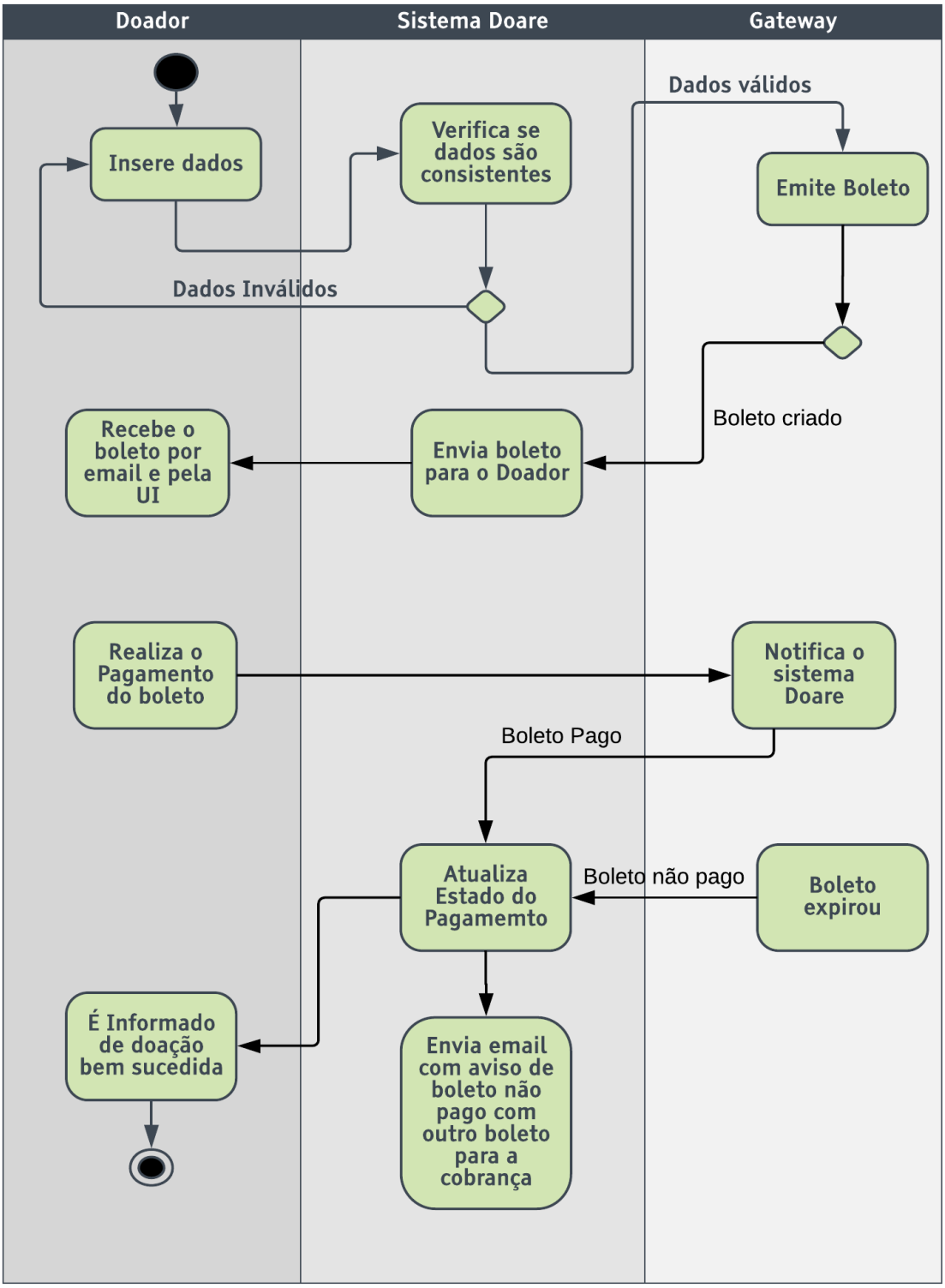

Figura 6 – Diagrama de atividade UC002

Nas Figuras 7 e 8 estão representados, respectivamente, o diagrama de sequência e o diagrama de atividade desse caso de uso.

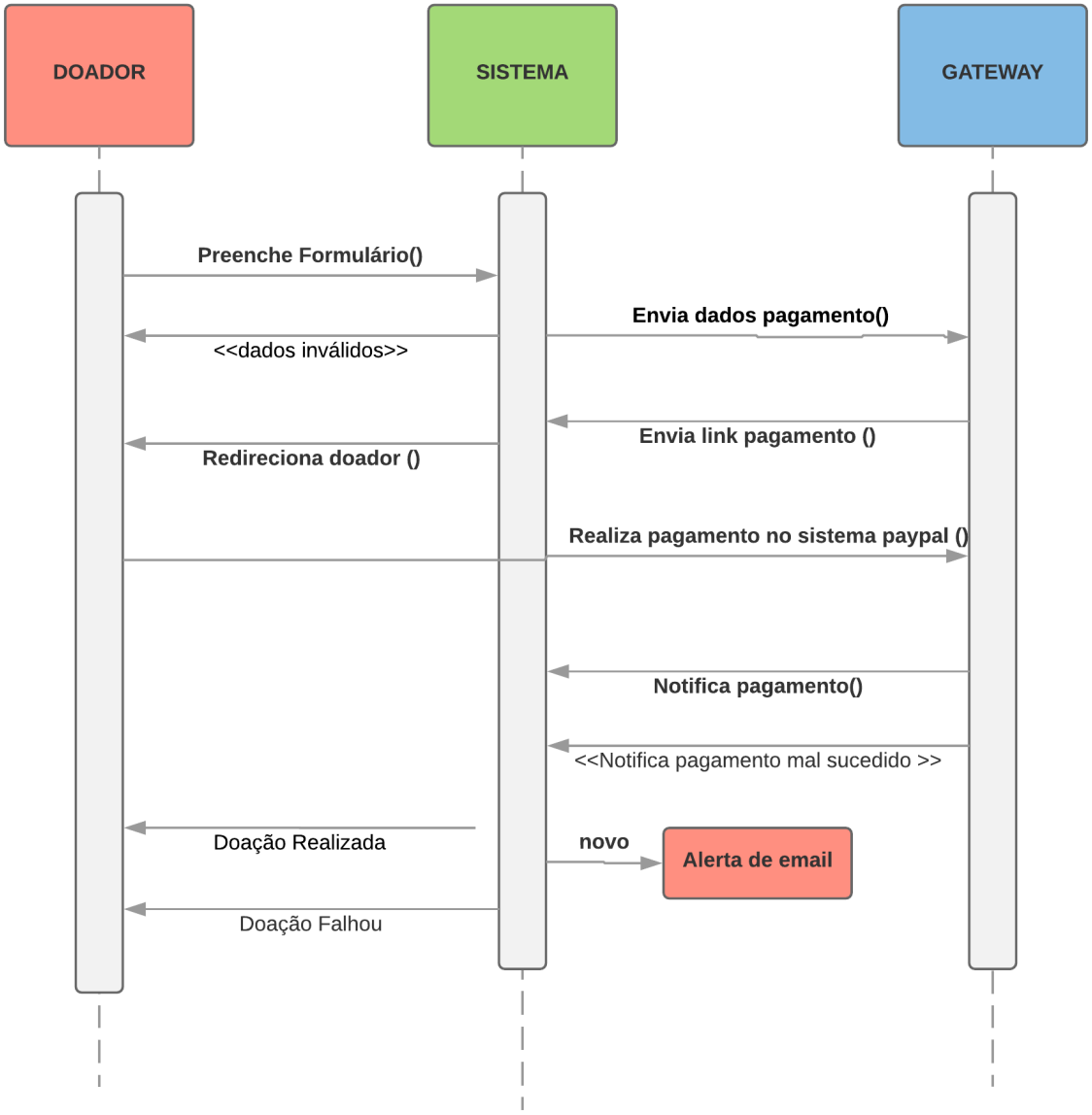

Figura 7 – Diagrama de sequência UC003

2.1.2 O painel do Administrador da ONG

O Painel da Doare é o espaço onde os administradores das ONGs atendidas pela Doare podem criar e editar campanhas, verificar o saldo arrecadado e informações sobre os seus apoiadores. No contexto deste trabalho foram feitas poucas contribuições para este sistema, uma vez que ele ainda está implementado no sistema legado da empresa, em PHP. As contribuições com essa sessão da base de código do sistema da empresa foge de alguma forma do contexto deste trabalho. Pequenos reparos foram realizados, alterações pontuais foram feitas, mas não houve um estudo aprofundado da arquitetura desta aplicação. Deu-se início, no entanto, próximo ao término da duração do estágio, o desenvolvimento de um novo painel, utilizando-se para isso as mesmas tecnologias utilizadas nos sistemas mais

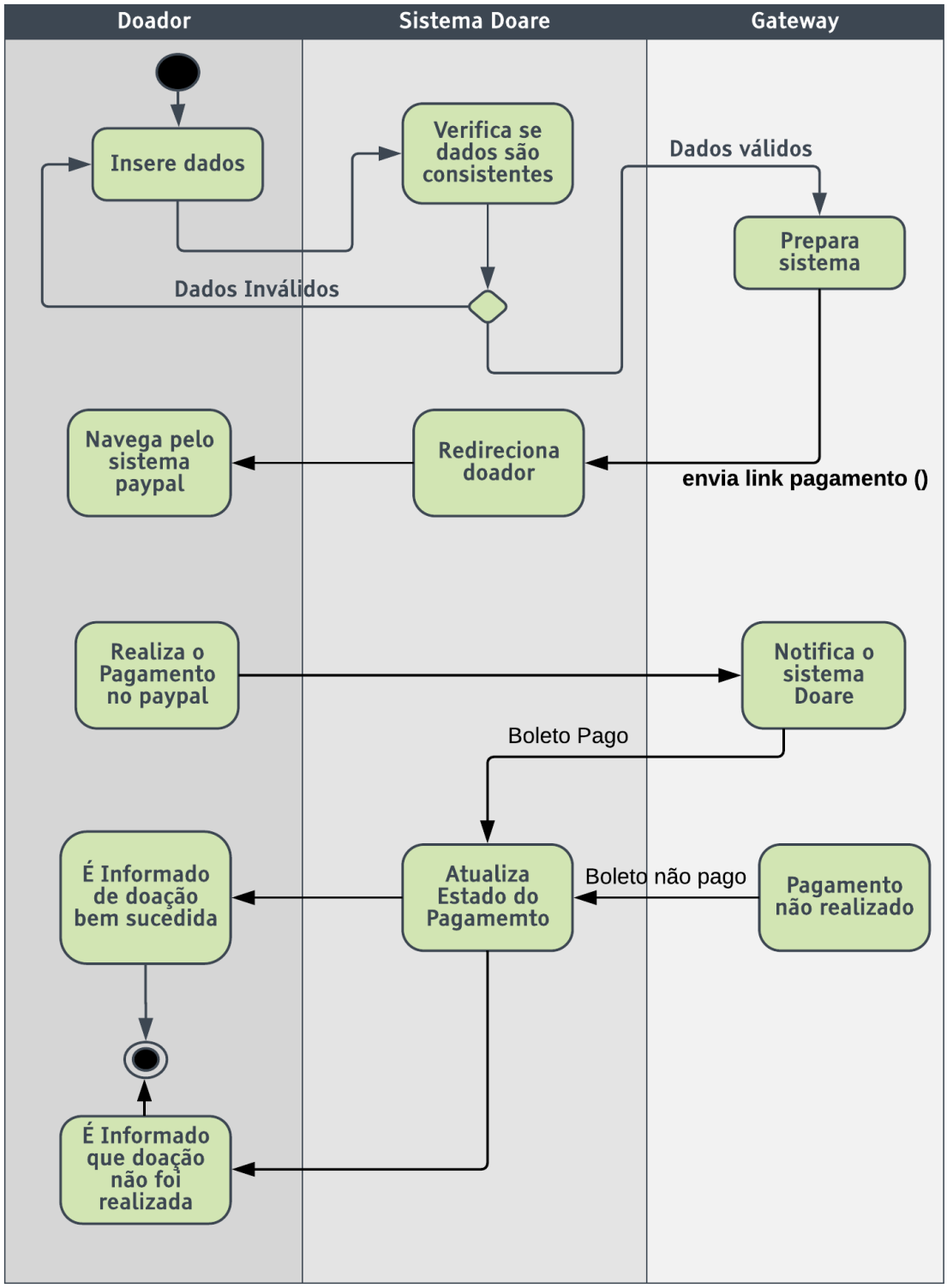

Figura 8 – Diagrama de atividade UC003

novos da empresa. Isso envolveu a preparação da API (que será descrita em detalhes no Capítulo 4 para atender as requisições deste novo painel.

2.1.3 Outras páginas administradas pela Doare

As outras páginas disponibilizadas pela Doare que integram seu sistema de doações são, em grande parte, páginas que usuários podem acessar para encontrar os produtos e atividades oferecidas pela empresa. Essas páginas, o portal da Doare, onde usuários podem encontrar perfis de ONGs para conhecê-las e então contribuir com elas, o portal Memio, onde, semelhantemente à página da Doare, doadores podem encontrar campanhas em forma de rifas nas quais doam e concorrem a uma gama ampla de experiências, e, por último, o portal Giveom, no qual usuários podem ter uma caixinha de doações, como a descrita anteriormente nesse capítulo, instalada em suas próprias páginas na *internet*.

A manutenção e aprimoramento desses sistemas não foi o foco das atividades desenvolvidas neste trabalho, ainda que algumas pequenas contribuições tenham sido realizadas.

2.2 Especificação da API

Uma API é, de maneira geral, o meio de comunicação que um programa de computador utiliza para se comunicar com outros programas. Uma série de métodos e protocolos de comunicação que permitem a interação entre programas que estão sendo executados em um computador ou em computadores diferentes definem uma interface de programa de aplicação, ou seja, uma API [4]

Na internet, um tipo muito comum de API são sistemas que são executados em servidores e que podem ser acessados por requisições HTTP. Esses programas funcionam como o cérebro das aplicações Web - neles é executada a parte lógica do negócio. Aplicações separadas que são responsáveis pela interação direta com o usuário trocam informações com essas APIs por meio dos métodos que ela oferece. Esses métodos são comumente chamados de *endpoints*, ou extremidades de uma API, uma vez que eles são seu ponto de contato com o mundo externo.

Nesta sessão serão detalhados os *endpoints* mais importantes da API da Doare - aqueles responsáveis pela criação de uma nova doação no sistema, e aqueles que possibilitam a criação e leitura de campanhas e organizações do sistema. Serão descritas as entradas e saídas existentes nesses *endpoints*. O padrão de comunicação utilizado para essa comunicação será descrito no Capitulo 4.

Na tabela da figura 9 estão representados alguns dos *endpoints* mais importantes da API da doare, com os os parâmetros de entrada esperados por eles e suas informações de resposta previstas.

	Parâmetros de entrada	Resposta esperada
Obter organizações	Limite, Offset (parâmentros de páginação)	Lista de organizações, cada uma com: nome, dados de contato total doado, link para logotipo, etc
Obter Campanhas	Limite, Offset (parâmentros de páginação)	Lista de campanhas, cada uma com: id, id da organização vinculada, nome, dados de contato, total doado, link para logotipo, etc
Obter organiazação	id da organização	Dados da organização: id, nome dados de contato, total doado, link para logotipo, etc
Obter campanha	id da campanha	Dados da campanha: id, id da organização vinculada, nome dados de contato, total doado, link para logotipo, etc
Criar organização	Dados da nova organização: nome, dados de contato, link para logotipo, etc	id da organização riada criada
Criar campanha	Dados da campanha: id da organização vinculada, nome dados de contato, total doado, link para logotipo, etc	id da campanha criada
Criar doação	Dados do doador: nome, email, cpf, dados de pagamento (valor, tipo de pagamento, dados especificos do meio de pagamento, como número do cartão), id da ong apoiada	Id da doação criada, situação da doação (bem sucedida, com erro...)
Obter doação	id da doação	Valor, situação, id da organização apoiada, id do doador

Figura 9 – *Endpoints* da API da Doare

3 Modelo de Solução - *Front-end* - Aplicação do usuário

Como na maioria dos sistemas *web* desenvolvidos na atualidade. O sistema descrito neste trabalho apresenta uma separação clara entre a aplicação que apresenta a interface do usuário, tipicamente chamada de *front-end*, ou UI, de *user interface*, e o *back-end*, a aplicação executada no servidor, que tem acesso aos bancos de dados, e é responsável por toda a lógica de negócio do sistema. Essas duas partes do sistema normalmente se comunicam por meio de requisições HTTP que serão discutidas no capítulo 4. Neste capítulo é descrita a solução implementada no *Front-End*. É discutido também como funciona o *Framework Angular*, que é utilizado como base para essa aplicação, e em seguida são detalhados alguns exemplos práticos da aplicação deste *framework* no sistema da Doare.

3.1 *Angular*

Tendo em vista tornar mais rápido o desenvolvimento de aplicações complexas para a web, um grupo de engenheiros de software da *Google* criou, em 2010, o *framework AngularJS*. Esta nova tecnologia permitia que interfaces gráficas de páginas da web pudessem ser construídas de maneira modular, sendo elas composições de vários componentes. Estes componentes, por sua vez, poderiam ser utilizados em várias páginas diferentes. Além disso, o *framework* permitia que o conteúdo das páginas seja atualizado de maneira dinâmica de acordo com o contexto no qual ele está sendo exibido. No sistema da Doare é utilizado o *framework* Angular, uma segunda versão do *AngularJS*, lançada em 2016 pelo mesmo time de desenvolvedores da *Google*. Nas próximas sessões serão discutidas brevemente as linguagens de programação (e outros tipos de linguagens) utilizadas para construir aplicações com o *Angular*, assim como os conceitos mais importantes do trabalho com a ferramenta. [5]

3.1.1 Linguagens utilizadas no *Angular*

Para o desenvolvimento de aplicações Web com o *Angular* são utilizadas, na verdade, linguagens muito semelhantes às linguagens HTML, *JavaScript*, e CSS, que são popularmente utilizadas para o desenvolvimento deste tipo de *software*. A maneira mais simples de construir um site é escrevendo uma página utilizando HTML (*Hypertext Markup Language*). Nesta linguagem constrói-se a estrutura da página. Nela estão escritos os conteúdos da página, junto a *tags* (etiquetas) que determinam se um pedaço de texto

é um cabeçalho ou um parágrafo, por exemplo. Uma página em HTML puro tem, no entanto, uma aparência bastante rudimentar. Para formatar o documento e dar a ele um estilo personalizado, surge a linguagem CSS (*Cascading Style Sheets*), que define como deve ser a aparência de cada elemento dentro dos arquivos HTML. Por fim, para dar dinamismo às páginas e construir aplicações que interagem com outros serviços e sistemas, a linguagem de programação *javascript* completa o trio, possibilitando a criação de *scripts* que descrevem o comportamento de um botão na página, por exemplo. No *angular*, e neste trabalho, no entanto, utiliza-se para cada uma dessas linguagens uma versão um pouco diferente. Estas versões serão descritas a seguir. [5]

3.1.1.1 *TypeScript*

O *framework Angular* é desenvolvido em *TypeScrit* e projetado para ser utilizado em*TypeScript*, uma linguagem de programação *open source* desenvolvida pela *Microsoft*. Trata-se de uma linguagem que quando compilada é transformada em *javascript*, sendo assim um superconjunto sintático dessa linguagem. A principal diferença entre o *TypeScript* e o *Javascript* é que o primeiro fornece orientação a objetos baseada em classes e tipagem dinâmica opcional, permitindo que se defina com mais clareza, por exemplo, que tipo de dados de entrada uma função deve receber, e que tipo de dados ela irá retornar. A vantagem dessa tipagem mais estática envolve uma maior robustez do código, já que a consistência da informação é testada de maneira mais intensa, e a possibilidade de criação de ferramentas de programação poderosas, como editores de texto e compiladores que identificam erros e mostram ao usuário de maneira mais eficaz. Isso tudo vem com o custo de tornar o desenvolvimento de novas funcionalidades um pouco mais trabalhoso e a linguagem menos flexível.

3.1.1.2 *EJS*

EJS,*Embedded JavaScript Templates*, ou modelos embarcados de *JavaScript* é uma linguagem de *template* que se parece bastante com código em HTML. No entanto, *templates* em EJS permitem que variáveis e funcionalidades de *JavaScript* sejam utilizadas para que um arquivo HTML puro seja gerado de maneira dinâmica. Com ela é possível, por exemplo, definir que uma parte da página vai aparecer ou não somente se uma condição externa for verdadeira, e até criar iterações em listas de forma a preencher informações na tela dinamicamente.

3.1.1.3 *SCSS*

Para este projeto foi escolhido, em vez da utilização do CSS como folha de estilos, a linguagem SCSS, *Sassy CSS*, que é uma variação mais próxima do CSS da linguagem SASS, *Syntactically Awesome Stylesheets*. Assim como o *TypeScript* é compilado para *JavaScript*

e o EJS é compilado para HTML, essas duas linguagens também são compiladas para CSS, oferecendo somente uma sintaxe mais fácil e ferramentas de edição melhores.

3.1.2 Visão Geral da Arquitetura do Angular

Antes de apresentar exemplos de como o *Angular* é utilizado para desenvolver a aplicação do usuário do sistema da Doare, convém detalhar como a estrutura padrão deste *framework* funciona e detalhar brevemente os principais conceitos que se precisa conhecer para trabalhar com ele. Para tanto optou-se por não entrar em questões muito aprofundadas, mas sim naquilo que é suficiente para compreender quais são os blocos que constroem uma aplicação em *Angular*, e porque a opção por trabalhar com essa tecnologia é bastante popular atualmente.

O *framework* oferece diversas bibliotecas que facilitam a criação de aplicações Web. Algumas dessas bibliotecas são essenciais em qualquer aplicação, outras são ferramentas opcionais. De maneira geral, primeiramente cria-se *templates* em HTML (uma versão de EJS especifica para o *Angular*). Em seguida escreve-se classes do tipo componente em *TypeScript* que realizam a parte lógica da aplicação e fornecem informações para os templates de maneira dinâmica, e que em contrapartida recebem informações dos templates quanto a eventos que ocorrem na interface, como o *click* em um botão ou preenchimento de um formulário. Toda a lógica que foge do escopo de um único componente e precisa ser acessada em diferentes partes da aplicação é construída em serviços. E tudo isso pode ser, conforme necessidade e tamanho da aplicação, colocado dentro de um módulo separado que contém os componentes e serviços de um determinado domínio da aplicação. Poderia haver, por exemplo, numa aplicação de rede social, um modulo para o histórico de notícias e outro para o serviços de mensagens privadas. É importante notar aqui que, quando se fala de lógica da aplicação no *front-end*, refere-se à lógica necessária para a execução da interface em si - a lógica de negocio e o acesso aos dados persistidos é responsabilidade do *back-end*. A seguir alguns desses conceitos serão elucidados com mais clareza.

3.1.2.1 Componentes

Componentes em uma aplicação *Angular* podem ser comparados a blocos de lego. Uma página vista por um usuário sempre é um componente, mas dentro dele podem haver vários outros componentes, cada um deles formando uma parte da página. Dá mesma forma, os componentes de uma página podem por sua vez conter outros componentes dentro de si. Com essa modularidade, o *Angular* permite que funcionalidades, envolvendo todo seu design e lógica, sejam desenvolvidas uma vez só e utilizadas de maneira flexível em diferentes partes da aplicação.

Um exemplo é bastante útil para exemplificar essa estrutura. Na fig 10 está representada uma página do *Angular* que serviria para mostrar uma lista de campanhas.

Esse componente poderia se chamar, por exemplo, página de campanhas. No entanto, deseja-se mostrar dentro dele componentes que são utilizados em diversas partes da aplicação, como o cabeçalho e rodapé da página, assim como um menu lateral. Esses componentes auxiliares são portanto incluídos dentro do componente da página de campanha, que incluí também a listagem.

Cada componente possuí propriedades, que podem ser acessadas de seu *template*, como a lista de campanhas, no exemplo da imagem. Além disso, os componentes possuem manipuladores de eventos, que agem de maneira adequada quando um evento é disparado no template.

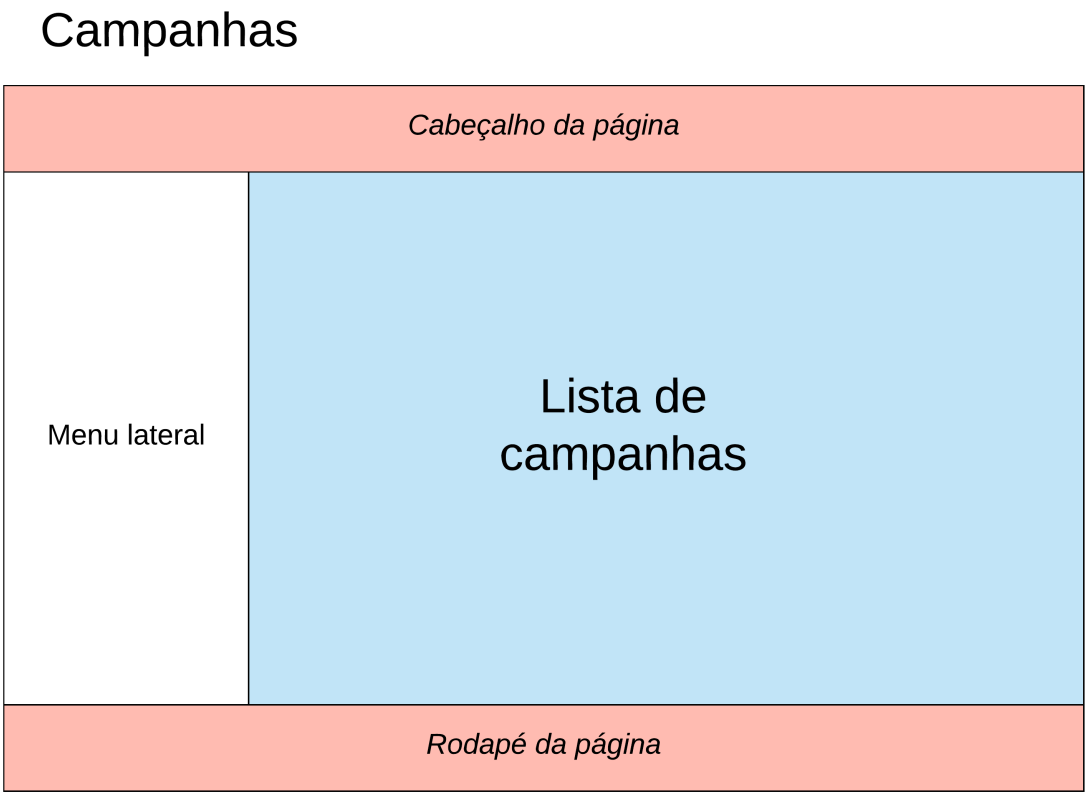

Figura 10 – Exemplo da estrutura de uma página em Angular

3.1.2.2 *Templates*

Nos *templates* é definida a estrutura visual dos componentes. Os textos e informações que serão mostrados, assim como a posição em que eles são mostrados. Os *templates*, como dito anteriormente, são responsáveis por disparar eventos que são administrados pelos seus componentes. Dentro dos *templates* também é definido onde outros componentes, comuns do HTML ou desenvolvidos no projeto, estarão localizados dentro do componente ao qual ele pertence, como o cabeçalho na figura 10. Um novo componente é inserido no *template* como se fosse um elemento HTML comum, como uma parágrafo ou título, além

disso pode-se passar parâmetros para o componente inserido, e inserir várias instâncias de um mesmo componente no *template*.

3.2 Exemplo prático - Como o sistema da Doare aplica a estrutura de componentes Angular

A caixinha de doações da Doare é um bom exemplo de como uma página da web pode ser montada a partir de vários componentes do *Angular*. Na figura 11 pode-se ver novamente o terminal de pagamentos, dessa vez com os componentes que a formam explicitados. A página em si é um componente, que contém alguns elementos, como o logotipo da organização e o valor do pagamento, mas também contém três componentes dentro dele. Um deles é o seletor de moedas e idioma, o outro um seletor de pagamentos, e o terceiro é um formulário especifico do meio de pagamento que está selecionado. Quando um outro meio de pagamento é selecionado, não há necessidade de a página inteira ser recarregada. Somente o componente do formulário é alterado para o do outro meio de pagamento, como pode-se ver na figura 12.

A vantagem de se organizar as páginas dessa maneira é que, além de recarregar somente uma parte da página quando ocorre uma mudança como essa, a parte do código que se refere a parte do terminal que não muda quando se altera o meio de pagamento não precisa estar presente em dois lugares diferentes do projeto. Tal ausência de repetição otimiza muito o tempo de desenvolvimento e manutenção da aplicação

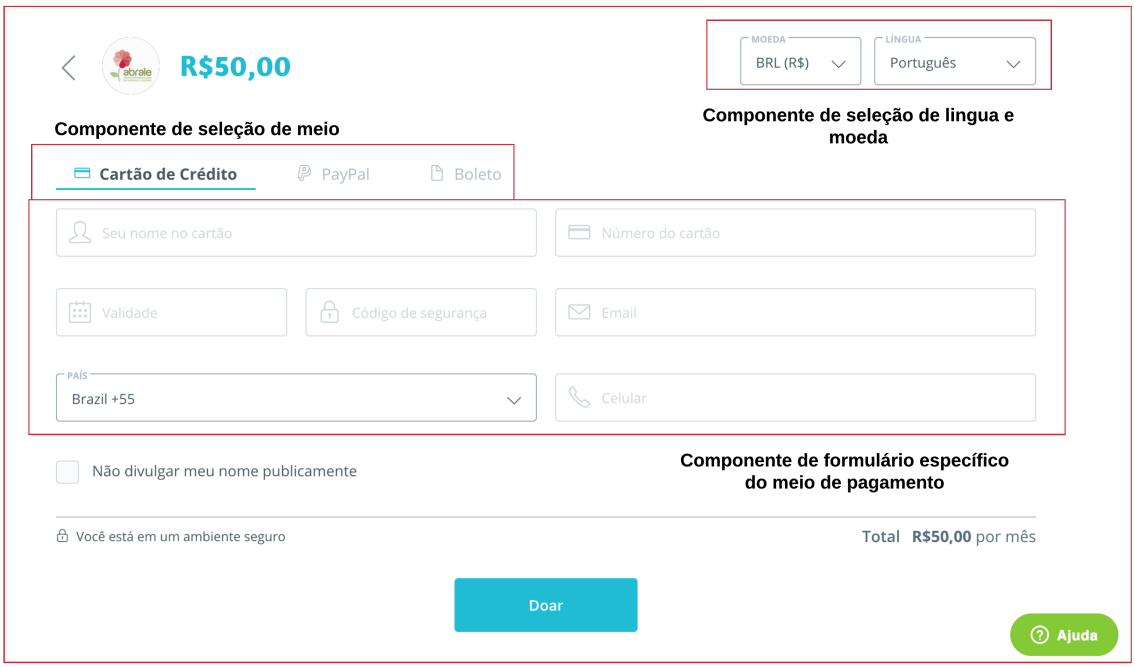

Figura 11 – Estrutura em angular da caixinha de doações - *Cartão de crédito*

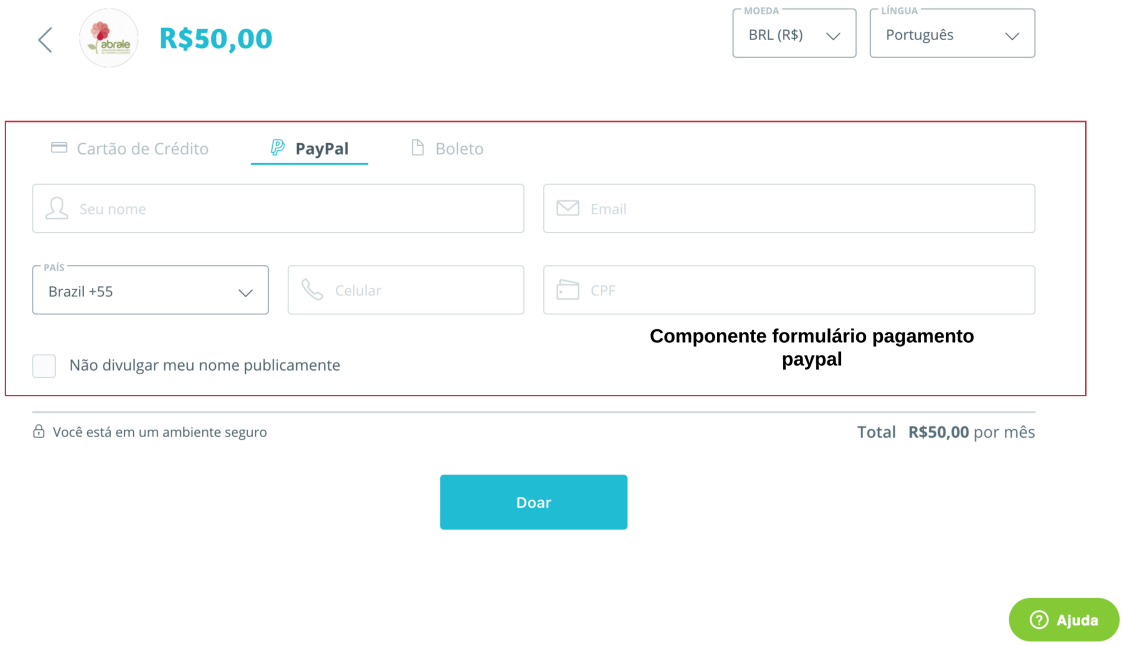

Figura 12 – Estrutura em angular da caixinha de doações - *Paypal*

4 Modelo de Solução - Back End - Arquitura da API

A aplicação de interface gráfica, executada no navegador web do usuário, se comunica continuamente com um sistema separado, executado em um servidor, no caso, a API (*Application Programming Interface*, ou interface de Programa de Aplicação) da Doare. A UI (*User Interface*, ou interface do usuário) requisita o servidor sempre que necessário para obter dados, assim como para enviar informações que devem ser processadas de acordo com a lógica de negocio do sistema. Esses dados trocados podem ser, por exemplo, listagens de campanhas que estão acontecendo no sistema, ou dados de pagamento para que uma doação seja realizada para uma dessas campanhas.

Neste capítulo será detalhado como se dá a comunicação entre a UI e a API, assim como os principais padrões de projeto que são utilizados nesse sistema. Primeiramente será discutido o conceito de microserviços. A API da Doare é dividida em algumas aplicações menores, que trocam informações mas têm responsabilidades diferentes e são executadas independentemente. Cada uma dessas aplicações, ou microserviços, tem como base estrutural uma arquitetura baseada em três conceitos, que serão discutidos. O DDD, ou Projeto orientado a Domínio, O CQRS, ou Separação de responsabilidades de Consulta e Comando, e o *Event Sourcing*. A combinação dessas três abordagens é utilizada frequentemente na industria para a construções de sistemas com nível de complexidade elevado.

A arquitetura do sistema da Doare descrita aqui não foi desenvolvida no escopo deste trabalho. No entanto, a construção de novas funcionalidades baseadas nessa arquitetura e manutenção do sistema como um todo exige a compreensão detalhada de como o sistema funciona e dos conceitos implementados nele. Boa parte dos esforços que originam este documento envolveram tais atividades de melhoria e manutenção, portanto a arquitetura implementada se torna relevante.

4.1 Os microserviços da Doare

Existe uma tendência na indústria de desenvolvimento de software, na qual empresas optam por desenvolver diversos pequenos sistemas que interagem entre si em vez de um único sistema muito grande que atende todas as necessidades do negócio. Essa abordagem, adotada por empresas grandes como o *Spotify* e o*Netflix*, permite que os recursos de desenvolvimento sejam direcionados, em cada momento, de maneira mais focada para cada problema a ser resolvido. [6]

Diferentemente de outras arquiteturas que serão mencionadas nesse capítulo, a ideia de microserviços não é tão especificamente definida. Cada empresa que escolhe essa abordagem a implementa de maneira diferente, ainda que seguindo a ideia de modularização e divisão de responsabilidades. O time de desenvolvimento de uma empresa pode escolher por criar dezenas de pequenos serviços ou poucos serviços maiores e a comunicação entre esses serviços podem se dar de diferentes formas, sendo que a mais comum delas, e também a utilizada no sistema da Doare, é o padrão de comunicação HTTP REST, que será descrito na próxima seção.

O desenvolvimento de vários sistemas separados traz uma maior complexidade do que a simples construção de uma aplicação monolítica responsável por todas as funcionalidades do software. Compreender como as diferentes aplicações interagem entre si e quais são as responsabilidades de cada uma delas demanda mais tempo de estudo de um desenvolvedor novo no projeto. No entanto, essa abordagem traz algumas vantagens. O sistema torna-se mais facilmente escalável, uma vez que pode-se direcionar recursos de desenvolvimento e processamento de maneira mais eficaz. Conforme a empresa ganha um maior porte, a equipe de desenvolvedores pode ser divida em várias pequenos times responsáveis pela totalidade de um microserviço ou de alguns microserviços, e para cada microserviço implementado pode-se utilizar tecnologias completamente diferentes, desde que se mantenha a língua franca de comunicação entre eles (como REST, no caso da Doare).

No sistema da Doare, até o momento a API é divida entre três microserviços. O mais central deles é o serviço Organizações. É com ele que a aplicação de *Front End* troca informações boa parte do tempo. Esse microserviço é responsável por tarefas relacionadas às organizações que utilizam o sistema da Doare, às campanhas realizadas por elas, assim como às Doações que elas recebem. O microserviço de organizações utiliza o segundo maior microserviço da empresa para processar a parte financeira das doações: O microserviço de pagamentos. Nessa parte do software que ocorre a comunicação com os *gateways* de pagamento e o registro e acompanhamento das transações realizadas. Um outro serviço, menor, é responsável pelo registro e autenticação de usuários do sistema. Na figura 15 está representado de maneira simplificada a estrutura dos microserviços da Doare e a forma como eles se comunicam.

4.2 A Comunicação entre a Interface Gráfica e os Microserviços da API

O meio de comunicação utilizado entre a aplicação do cliente e o servidor é o protocolo HTTP (*Hypertext Transfer Protocol*), um protocolo de aplicação baseado nos protocolos TCP/IP que é o mesmo conjunto de regras de transferências de dados pela

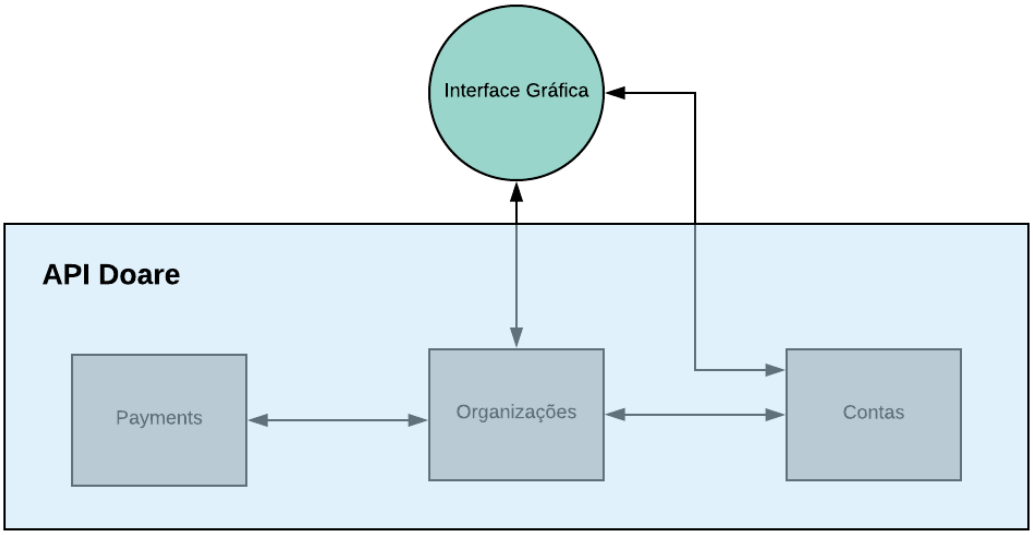

Figura 13 – Microserviços da API Doare

internet utilizado quando um usuário digita o endereço de uma página da internet em seu navegador. No caso das requisições HTTP entre a UI e a API, utiliza-se a estrutura REST(*Representational State Transfer*, ou transferência de estado representacional). REST define uma maneira de enviar requisições HTTP baseadas em um número limitado de métodos, que podem representar a inserção de um novo registro no sistema (com o método POST), a atualização de um existente (com o método PATCH), ou a simples requisição de informações de registros (com GET). As requisições REST seguem uma estratégia comumente chamada de *Stateless*, ou sem estado, na qual cada requisição é tratada como uma tarefa isolada pelo servidor. Isso é importante para que o sistema não dependa de uma comunicação continua entre a API e a UI. Um caso que ilustra a importância disso é a situação na qual a API está sendo executada em dois servidores diferentes. A UI pode requisitar informações sobre uma campanha de um servidor, e em seguida enviar os dados de pagamento para o outro servidor. Como as requisições são independentes e sem estado, isso não causaria problemas, e a operação seria realizada normalmente. [7]

4.3 Domain Driven Design

O *Domain Driven Design*, projeto orientado a domínio, ou DDD, é uma abordagem de desenvolvimento de software baseada na concepção de que cada aplicação que se deseja construir pertence a um domínio. O domínio é definido pelo assunto do qual a aplicação trata e da forma como ela deve se comportar, o que é usualmente chamado de lógica de negócio. Toda a esfera de conhecimento e atividade que deve resultar na lógica de uma aplicação define, portanto, seu domínio. [8]

O termo *Domain Driven Design* foi cunhado em 2004 por Eric Evans em seu livro homônimo. No desenvolvimento do sistema da Doare busca-se implementar essa filosofia de projeto. Portanto, nesta sessão busca-se aprensentar de maneira sucinta alguns dos conceitos que são abordados do livro, elucidando assim a ideia por trás do DDD, e junto a isso são colocados exemplos de como o Domínio da Doare é representado nesse tipo de modelagem.

4.3.1 O Domínio

O conceito de Domínio é uma ferramenta muito útil para o desenvolvimento de software. Quando se concebe o modelo do domínio da aplicação, consegue-se traduzir de maneira clara as necessidades reais do usuário em uma linguagem que pode ser traduzida sistematicamente em lógica de programação. O que torna isso possível é que o projeto de uma aplicação orientada a domínio busca implementar a lógica de negócio da aplicação de maneira que ela se asemelhe o máximo possível com a forma como especialistas do domínio o compreendem e descrevem.

O modelo do domínio da aplicação define, portanto, uma linguagem franca, um léxico que deve ser utilizado desde a concepção de como a aplicação será utilizada pelo usuário final até a escrita do seu código fonte. Eric Evans chama essa linguagem de ubíqua e na Figura 14 está representado qual parte dos jargões de um time fazem parte dela. Em um cenário onde os especialistas de negócio sabem pouco sobre programação e os desenvolvedores sabem pouco sobre o problema a ser resolvido com o software, o estabelecimento de termos e conceitos comuns possibilita que os problemas que o software visa resolver sejam abordados de maneira precisa e eficiente e ajuda a evitar divergências entre a lógica implementada e aquilo que é idealizado pelo especialista de domínio.

4.3.2 Isolando o Domínio

O Software desenvolvido de maneira guiada pelo modelo do domínio deve possuir uma parte de sua implementação que expressa da forma mais próxima possível esse modelo. Essa parte da aplicação normalmente é pequena em termos de linhas de código ou tempo de trabalho, mas pode-se dizer que é a parte central da aplicação. Um fator importante, é que essa parte deve ser facilmente reconhecida dentro do sistema. Como Eric Evans coloca em seu livro [8], não deve ser necessário procura-la na aplicacão em meio a uma mistura de objetos não relacionados a ela, como se estivessemos procurando constelações em um céu estrelado. O Domínio deve, portanto, estar isolado do resto do software em uma sessão onde ele seja expressado de maneira independente de aspectos técnicos quanto à infraestrutura da aplicação ou à interface gráfica, por exemplo.

Para garantir-se o isolamento do Domínio, projetos que seguem a metodologia

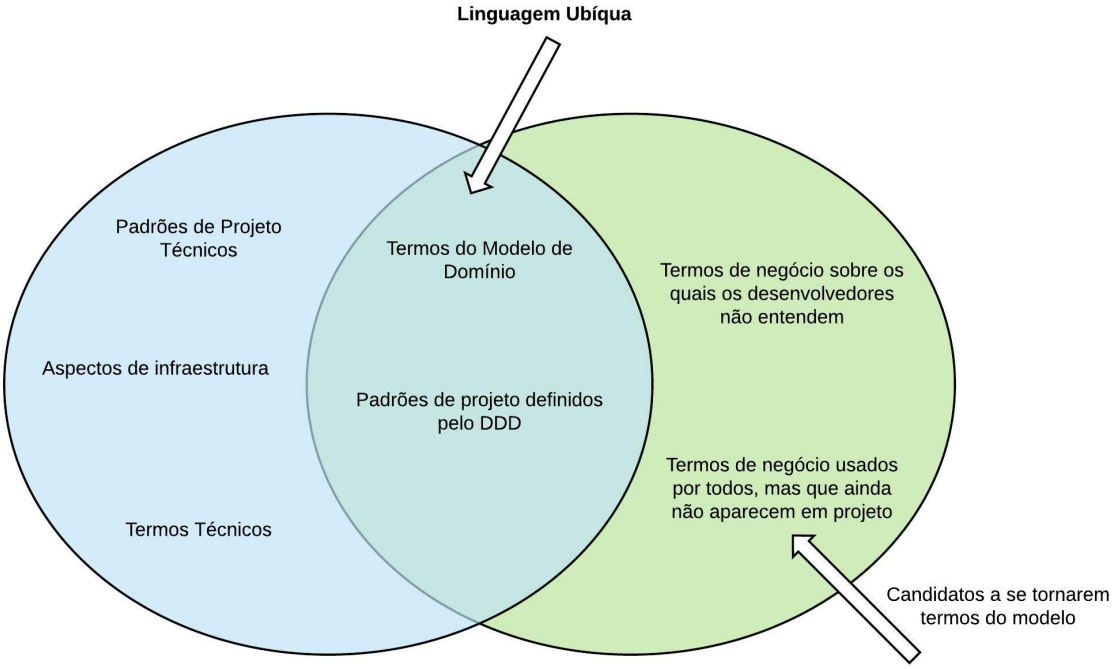

Figura 14 – Linguagem Ubíqua - Figura adaptada de Evans (2016)

DDD costumam implementar uma arquitetura de software em camadas, o que é uma pratica comum inclusive em projetos que não utilizam DDD. A ideia da arquitetura em camadas é que cada camada realize funções específicas dentro do software, e que camadas superiores dependam somente de camadas abaixo delas. No DDD, o domínio é uma dessas camadas. Na Figura 15 estão representadas as camadas que normalmente são implementas em um projeto guiado a domínio.

No caso do sistema da Doare, quando um usuário realiza uma doação, por exemplo, é necessário que exista uma parte do código da aplicação que desenhe o terminal de doações na tela para o usuário (Interface Gráfica), outra que comunica a interface gráfica com as camadas de lógica (aplicação), validando e interpretando informações, outra que cria uma nova doação, e associa ela a um Doador, a uma organização, e a uma transação (Domínio), e uma camada que é responsável por persistir essas informações em um banco de dados (Infraestrutura).

A Camada de aplicação é, portanto, responsável por mostrar informações do usuário e interpretar seus comandos. A camada de aplicação, por sua vez, é uma camada fina define as ações que o software pode executar, mas sem expressar aspectos de lógica e conhecimento de negócio. A camada de domínio deve então representar os conceitos de negócio - informações sobre a situação dele, e suas regras. Aspectos mais técnicos, que dão suporte às camadas superiores, como a armazenagem e recuperação de dados do banco de

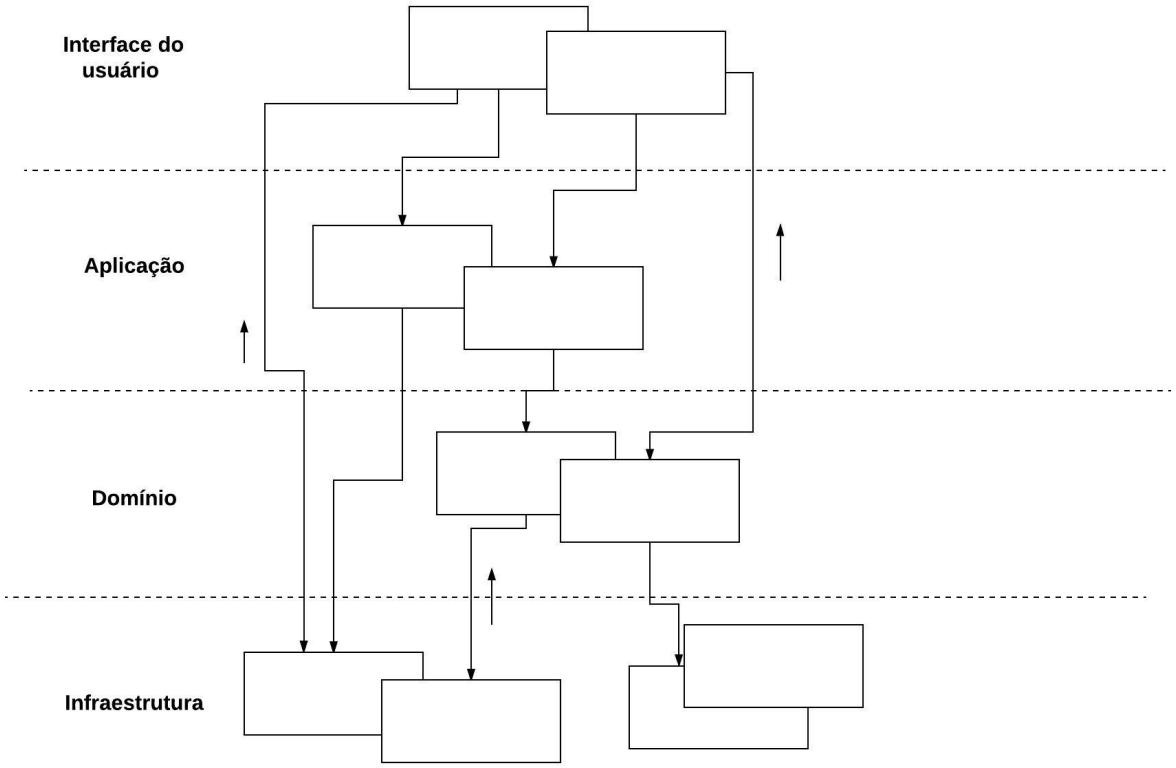

Figura 15 – Arquitetura em Camadas - Figura adaptada de Evans, Eric - Domain Driven Design, 2016

dados, sistemas de roteamento do servidor, ou a capacidade de mostrar informações no monitor do usuário e receber os comandos de seu mouse, teclado, ou tela de celular devem ser deixados fora dessa camada.

4.3.3 Transformando o modelo do domínio em Software

Para que o domínio esteja presente em uma camada isolada da aplicação, os conceitos e lógicas que o definem devem permear a solução implementada em detalhe. Enquanto que desenhar diagramas e pensar nesses elementos de maneira abstrata seja relativamente simples, expressa-los de maneira programática pode revelar-se um grande desafio.Eric Evans propõe em *Domain Driven Design* elementos abstratos que servem como interface entre o domínio que está no papel e o programa em sí. Nesta subsessão serão discutidos brevemente os principais deles, junto a exemplos de como essas abstrações foram implementadas no sistema da Doare.

4.3.3.1 Entidades

Talvez a abstração mais importante do DDD - para explicar o conceito torna-se interessante passar por um pouco de filosofia. Pode-se fazer a pergunta: o que define a identidade de um indivíduo? Algumas pessoas argumentariam que um indivíduo humano

é representado por suas características - cor do cabelo, sua altura, seu nome e as feições de seu rosto. Essa resposta é, no entanto, facilmente refutável. Pode-se pintar o cabelo, crescer, e, hoje em dia, até mudar de nome, mas ainda assim, considera-se que um idoso possui a mesma identidade da criança que nasceu décadas atrás.

Uma entidade para o DDD é, portanto, um objeto que não é definido por suas características, ou atributos, e sim pela unicidade de sua identidade. Trata-se de um objeto do sistema que pode ser alterado com o passar do tempo, mas que mantem sempre a sua identidade. Enquanto que o exemplo mais intuitivo de uma entidade seja um ser humano, ou então o usuário de um sistema, modela-se de fato em DDD entidades como tudo aquilo que deve manter uma continuidade durante todo o funcionamento de um sistema. No sistema da Doare, por exemplo, cada Organização que recebe doações pela plataforma é uma entidade, assim como cada doador, cada doação e assim por diante.

Além das características de um objeto, também são modeladas nas entidades as funcionalidades daquilo que elas representam, ou seja, aquilo que elas podem fazer e aquilo que pode ser feito com elas. Um doador pode se registrar no sistema ou realizar uma doação. Uma doação, por sua vez, pode ser criada, paga, ou estornada, enquanto que uma organização pode receber uma doação, sacar sua arrecadação ou editar seu perfil público. Na implementação da entidade essas funcionalidades, ou métodos, aparecem de maneira a expressarem mais claramente o que eles significam, direcionando o resto da aplicação a fazer o resto do trabalho, sem que detalhes mais técnicos da implementação apareçam.

O conceito de entidade é importante para que se mantenha a consistência das informações no sistema desenvolvido, e até entre diferentes sistemas desenvolvidos. Uma mesma pessoa pode se registrar em dois sistemas separados de uma empresa, mas ela precisa ser identificada pelos dois sistema como uma só, possuir a mesma entidade. Isso poderia ser realizado, nesse caso, levando o email da pessoa como fonte de verdade quanto à identidade de um usuário. Muitas vezes essa identificação única das entidades se dá por um número ou código de identificação.

4.3.3.2 Objetos de Valor

Nem todos os objetos que possuem atributos no modelo de um domínio possuem uma identidade. Alguns objetos servem simplesmente para representar características de alguma coisa. Quando compra-se um computador em uma loja e ele apresenta defeito poucos dias depois, troca-se o computador por um outro idêntico. Se o novo computador funcionar bem, o trabalho pode continuar sem problemas. A identidade do computador não importa nesse caso, mas sim as suas características. Ele pode ter uma capacidade grande de memória e ter um design minimalista, mas não possui necessariamente uma identidade que o defina.

Um computador ou um produto qualquer poderá ter um número de série que o

torne único, mas no modelo de um domínio que só se importa com suas características esse número de série torna-se irrelevante. Esse tipo de objeto é modelado em DDD, portanto, como um objeto de valor. São objetos descritivos do domínio, dos quais não importa quem ou quais eles são, mas sim o que eles são.

Eric Evans traz em *Domain Driven Design* um exemplo interessante para ilustrar a diferenciação entre objetos de valor e entidades. Dependendo do sistema implementado, o mesmo objeto do mundo real pode ser melhor representado como uma entidade ou um objeto de valor. No sistema de um serviço de vendas pela internet, por exemplo, quando uma compra é realizada, o cliente preenche um endereço para o qual a encomenda deve ser enviada. Se o irmão do cliente compra outro produto na mesma loja e preenche o mesmo endereço, o sistema não precisa saber que os endereços são o de fato o mesmo. Nesse caso, endereço é um objeto de valor. Para o sistema dos correios, no entanto, cada residencia pode ter um registro de entregas realizadas, e as entregas para o mesmo endereço precisam ser feitas juntas, por razões óbvias de otimização de recursos. Nesse outro sistema, endereços são modelados como entidades.

À primeira vista a ideia de objetos de valor não parece trazer nada de muito novo em relação às entidades. De fato, seria possível implementar todos os objetos expressados no modelo como entidades, mas isso seria bastante caro em termos de desempenho, e tornaria a implementação menos coerente com o modelo. A infraestrutura para administrar entidades, mesmo com os bancos de dados modernos e otimizados disponíveis hoje em dia, é cara relativamente lenta. Objetos de valor são uma forma de representar objetos sem que seja necessário se preocupar com isso. Eles podem ser parte integral na criação de uma entidades, ou ainda objetos transientes dentro do programa, que nunca são individualmente persistidos em um banco de dados, sendo criados para uma operação e depois descartados.

No sistema da Doare, um objeto de valor importante implementado é o objeto que representa um cartão de crédito. Por razões de segurança, informações dos cartões de créditos utilizados para doações não são guardadas no banco de dados. Essas informações passam pelo sistema sendo instanciadas como o objeto de valor cartão de crédito. O sistema leva então essas informações para um *gateway*, que processa o pagamento e envia uma resposta que informa se a transação foi realizada com sucesso ou não. Nesse ponto, os dados do cartão de crédito já foram descartados, e o sistema só se importa com o resultado da transação. Mesmo quando o sistema precisa realizar cobranças periódicas no mesmo cartão de crédito, como ocorre no caso de doações recorrentes, os dados do cartão não são guardados no sistema. O que ocorre nesse caso é que o gateway de pagamento fornece ao sistema uma chave que pode ser utilizada pela Doare para realizar cobranças nesse cartão de crédito. Essa chave só funciona para cobranças realizadas pela Doare, que com ela realiza transações sem saber os dados do cartão. Dessa forma o cartão de crédito fica protegido de sistemas maliciosos que poderiam roubar esses dados.

Saber diferenciar entre objetos de valor e entidades de um domínio pode ser de grande valia para obter-se uma implementação concisa da lógica do domínio, podendo-se dessa forma melhorar a performance do sistema e diminuir a complexidade de sua manutenção.

4.3.3.3 Serviços

Ainda que o DDD seja uma técnica de desenvolvimento totalmente pensada em objetos, algumas das características de um domínio não podem ser diretamente associadas a um objeto específico. Ainda assim, para continuar com as abstrações de objetos, quando existe algum tipo de funcionalidade que não pertence a nenhuma das entidades ou a nenhum dos objetos de valor modelado cria-se um serviço que expresse e forneças esse comportamento ao domínio.

É importante notar que nem toda funcionalidade de um domínio deve ser mantida fora das entidades e objetos de valor. Pode parecer tentador colocar tudo em serviços separados, o que roubaria do DDD um pouco de seu sentido. No entanto, o excesso de complexidade não relacionada a uma entidade dentro dela pode acabar ofuscando aquilo pelo o que a entidade é realmente responsável.

Serviços são, portanto, a abstração de um tipo de objeto que é muito mais importante por aquilo que ele é capaz de fazer por outros objetos do que por suas características. Eles são muito úteis para guardar operações que são compartilhadas por diferentes entidades e objetos de valor dentro do sistema, operações que orquestram ações entre vários objetos diferentes, ou ainda operações que se comunicam com fatores externos ao domínio. Eric Evans propõem em *Domain Driven Design* três características que definem um bom serviço:

- A operação se relaciona com um conceito do domínio que não é uma parte natural de uma entidade ou objeto de valor.
- A operação é definida em termos de elementos do modelo do domínio
- A operação não tem estado (Stateless).

Uma operação sem estado significa aqui que ela é executada em uma instância única cada vez que o serviço é requisitado, sem guardar um histórico de requisições anteriores. O serviço pode ter acesso a informações globais da aplicações e causar mudanças nelas, mas ele não é capaz de mudar parâmetros que influenciem seu próprio comportamento.

No sistema da Doare, uma aplicação interessante do conceito de serviços se dá na implementação do sistema de pagamentos. A principal entidade que é envolvida em um pagamento é a entidade Transação. Ela guarda informações quanto ao valor da transação,

sua data, e seu andamento, ou seja, se está aguardando pagamento, se está paga ou se algum problema ocorreu. A abstração da entidade também oferece ações como confirmar a transação ou reembolsá-la. No entanto, o contato com os *gateways* de pagamento para os diferentes métodos de pagamento disponíveis é algo que não pertence propriamente à entidade transação. Essa funcionalidade é portanto colocada em um serviço. De fato, a implementação oferece diferentes serviços que podem ser utilizados pela entidade da transação, cada um deles sendo capaz de processar pagamentos de tipos diferentes (boleto, cartão de crédito, transferência, ou outros), utilizando *gateways* de pagamento diferentes (Moip, Stripe, Pagar-me, entre outros). Dessa forma a entidade da transação não precisa se preocupar com a forma como o pagamento é processado. Ela simplesmente instância o serviço correto para o tipo de pagamento que ela possuí, e esse serviço faz todo o processamento como necessário. A comunicação da transação com cada um desses serviços se dá de maneira padrão, na qual o processamento de um pagamento é solicitado para o serviço com determinados parâmetros, e como resposta à requisição o serviço informa se o pagamento obteve sucesso, junto a outras informações relevantes.

4.4 CQRS e Event Sourcing

Outros dois conceitos que frequentemente aparecem na literatura em conjunto com o *Domain Driven Design* é o de *Event Sourcing*, que basicamente idealiza o armazenamento de informações de um sistema baseado em eventos que ocorrem desde o inicio de sua utilização em vez de seus simples estado atual, e CQRS *Command Query Responsability Segregation*, ou Segregação de Responsabilidade de Comando e Consulta, que parte do principio que os processos de guardar informação no sistema e recupera-la devem ser abordados separadamente, com estruturas diferentes sendo responsáveis para cada uma dessas tarefas. Esses dois conceitos são independentes um do outro, mas se complementam de maneira muito favorável. A responsabilidade pela popularidade dessa combinação de padrões de desenvolvimento é Greg Young, que escreveu diversos artigos e deu muitas palestras sobre o assunto. Esta sessão é largamente baseada neste material. Nesta sessão serão elucidados brevemente esses dois conceitos. [9]

4.4.1 CQRS

De maneira simplificada, pode-se categorizar as ações realizadas em um sistema entre aquelas que modificam o estado do sistema e aquelas que simplesmente fornecem uma informação. Em alguns sistemas podem existir ações que se encaixam em ambas as categorias, mas em seu livro *Object Oriented Software Construction* [10], Betrand Meyer introduz o conceito de *Command Query Separation* (Separação de comando e Consulta), ou CQS que é a ideia de que, para se ter clareza entre aquilo que muda o estado do sistema e aquilo que não muda, os métodos dos objetos de um sistema devem seguir essa separação,

ou seja, consultas retornam informações e não mudam o estado do sistema, enquanto que comandos mudam o estado do sistema mas não retornam nenhuma informação. [1]

O conceito de *Command Query Responsability Segregation*, ou CQRS, é uma extensão do mesmo conceito. A diferença aqui é que essa separação ocorre de maneira ainda mais significativa. Toda a responsabilidade do sistema em realizar consultas e comandos fica separada. Isso significa que onde antes havia somente um objeto que possuía métodos desses dois tipo, agora há dois, implementados frequentemente com arquiteturas totalmente diferentes, cada um deles especializado em uma das duas operações.

Uma das principais vantagens dessa abordagem são o fato de que ela proporciona uma maior modularidade e flexibilidade para o sistema, tornando possível que novos métodos de consulta, que costumam ser mais numerosos que os de comando, sejam criados sem que exista a preocupação de causar algum efeito colateral no estado da aplicação, e que métodos que de fato alteram informações do sistema estejam separados do resto de maneira bastante explicita, podendo ser criados e alterados com maior clareza. Outro benefício de CQRS é que ela favorece muito a escalabilidade do sistema, ou seja, sua capacidade de crescer. Na maioria dos caso, um sistema realiza muito mais consultas do que comandos. Por outro lado, os comandos necessitam de mais processamento, e demoram mais para ser realizados. Em um sistema onde essas duas responsabilidades estão separadas, seria possível, por exemplo, dar mais poder de processamento para cada instância da parte do sistema que modifica o seu estado, e por outro lado criar mais instâncias da parte do sistema que realiza consultas, que é mais leve, porém mais requisitada.

Não se recomenda, no entanto, que o CQRS seja uma regra absoluta no desenvolvimento de um sistema. Em alguns casos, onde os objetos são muito simples e não centrais para o negócio, o desenvolvimento de um sistema que siga o padrão CQRS pode criar uma camada de complexidade sem trazer necessariamente benefícios reais. A implementação do CQRS deve ser, portanto, avaliada para cada aspecto do sistema desenvolvido.

Na Figura 16 está representado um esquemático de como uma arquitetura baseada em CQRS pode funcionar. A leitura e escrita de dados ocorre por vias separadas dentro do fluxo da aplicação.

4.4.2 Event Sourcing

Perguntando-se para as maiorias das pessoas como elas acreditam que os dados de um sistema devem ser armazenados, é bem provável a resposta seria algo próximo de um banco de dados relacional, onde as características atuais dos objetos do sistema estão guardadas. Se observarmos outras atividades profissionais, mais maduras, como coloca Greg Young, informações importantes são guardadas de maneira bem diferente. Um contador, quando paga uma conta, não apaga todas as transações que foram realizadas até

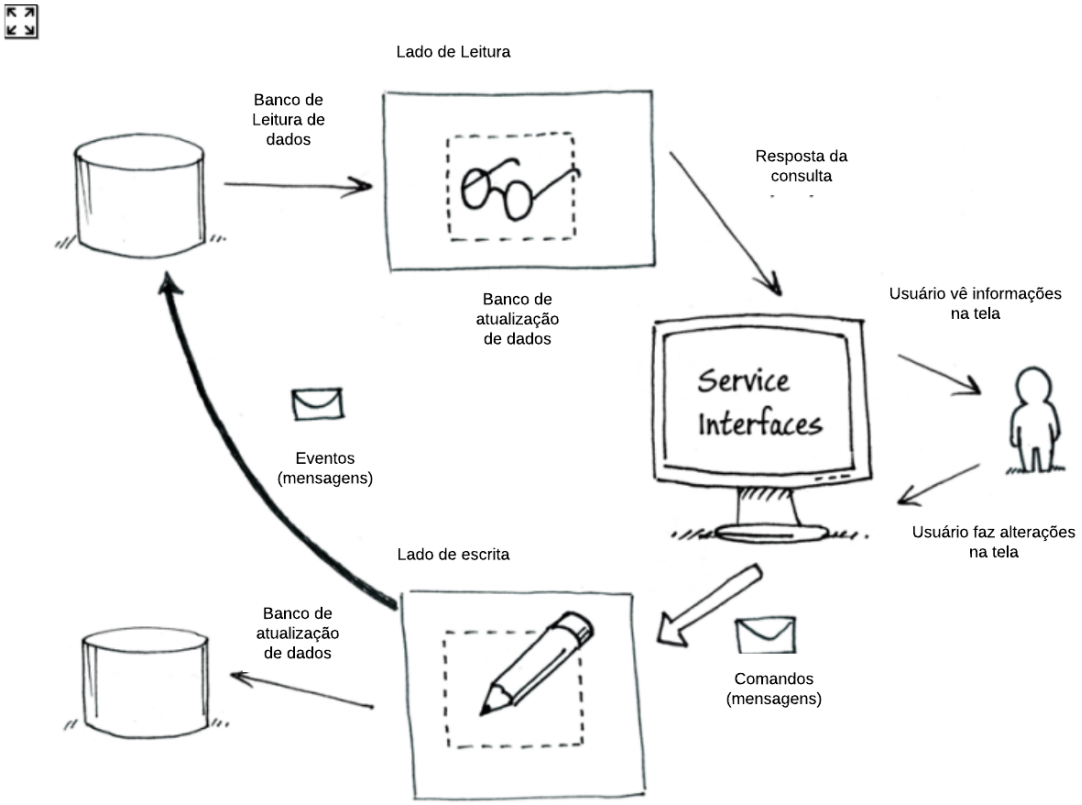

Figura 16 – CQRS - Microsoft - Figura adaptada de *Introducing the Command Query Responsibility Segregation Pattern* [1]

o momento para anotar o novo saldo. Ele simplesmente registra a transação realizada em uma lista de transações. Algo parecido acontece numa conta bancária, já que um banco tem que ter o registro de cada transação que ocorreu na conta para poder justificar o saldo atual apresentado. Saindo das transações financeiras, um médico quando vê um paciente antigo também não apaga todos os registros anteriores de sua saúde. Ele adiciona um novo registro a um histórico do paciente, para assim ter uma melhor compreensão do que ocorre com ele ao passar dos anos. Essa ideia de guardar informações por meio de registros do que ocorre com elas é a ideia por trás do Event Sourcing.

Event Sourcing, portanto, é uma forma de persistir o estado de uma aplicação armazenando o histórico que determina esse estado. Um exemplo em *Introducing Event Sourcing*, da *Microsoft*, [11] elucida bem a diferença de abordagem proposta com o *Event Sourcing*. Um sistema de administração de conferências precisa saber o numero de ingressos vendidos para saber se ainda há lugares vagos na hora de uma venda. O sistema poderia guardar o número de ingressos vendidos de duas maneiras. Ele poderia armazenar o número de ingressos vendidos e adicionar um a esse número toda vez que um ingresso é vendido, ou então ele poderia armazenar todas as vezes que um ingresso de uma conferência foi

vendido ou uma venda cancelada. Nesse caso, antes de permitir uma nova compra, o sistema passaria por todas as vendas para determinar se o número de ingressos vendidos já alcançou o limite disponível.

A principal vantagem, e também a mais evidente, de armazenar informações dessa maneira é a de que nenhuma informação é perdida, em nenhum momento, já que trata-se de um sistema onde informações são somente adicionadas (*append only*). Essa característica pode ser de grande valia para auditorias, e além disso informações úteis sobre o negócio podem ser obtidas dos eventos. Do histórico de um paciente um médico pode observar que ele costuma apresentar sintomas em determinada época do ano e em outras não, e no sistema de venda de ingressos poderiam ser determinados os funcionários que mais venderam ingressos. Um pequeno revés da utilização de *Event Sourcing* é exatamente aquilo que o define: Não se pode alterar informações, somente adiciona-las. Isso pode se tornar um problema quando um programador comete um erro e faz com que informação equivocada seja adicionada, e modificações na estrutura do sistema podem se tornar mais complicadas, já que é necessário que o novo sistema seja capaz de lidar com eventos que foram gerados por versões anteriores. Novamente, nesse caso, Greg Young se volta a industrias mais maduras para encontrar a solução. Um contador ou um médico já tiveram que lidar com mudanças no contexto em que trabalhavam sem descartar informações antigas, ou quando cometem um erro de registro. O mesmo pode ser realizada quando se desenvolve software com *Event Sourcing*. Um erro de registro pode ser reparado com um novo evento de correção, e um sistema com novos eventos não deve deixar de ser capaz interpretar os eventos já existentes.

O sistema da Doare lida com transações financeira, o que o torna crítico. Neste contexto, a utilização do armazenamento por evento traz uma maior confiabilidade para as informações presentes no banco de dados, assim como um histórico mais rico das transações que pode ser utilizado para estatísticas e analises de desempenho.

Existem diferentes maneiras de implementar o armazenamento por *event sourcing* em um sistema. A primeira maneira em que talvez se pensaria seria o armazenamento dos eventos utilizando-se um banco de dados relacional. No entanto, existem ferramentas especializadas que otimizam a escrita e leitura de eventos de uma implementação do *Event Sourcing*. Essas ferramentas garantem a consistências dos eventos adicionados, e oferecem uma infraestrutwura que auxilia no cumprimento dos *Event Sourcing*, como a característica de ser uma forma de armazenamento onde somente se adiciona, e nunca se remove informação. Na doare optou-se por utilizar a ferramenta de código aberto*Event Store*, um banco de dados funcional especializado em event sourcing desenvolvido pelo próprio Greg Young.

4.5 DDD, CQRS e Event Sourcing em conjunto e sua implementação no sistema da Doare

Os três conceitos que acabam de ser apresentados aparecem frequentemente em conjunto em diversos contextos, e no sistema da Doare eles constituem a base da arquitetura da nova API da empresa, sobre a qual uma parte significativa deste trabalho foi baseado. Buscou-se, portanto, no desenvolvimento do sistema, modelar o negócio da empresa utilizando-se a abordagem do *Domain Driven Design*, identificando quais são as entidades, objetos de valor e serviços do Domínio trabalhado, implementar *event sourcing* utilizando os eventos relevantes de cada uma das entidades modeladas, e separar a leitura das informações do sistema de sua escrita como determinado pelo conceito de CQRS. Nesta sessão será apresentado em maior detalhe como se dá tal implementação como um todo.

É importante relembrar que o presente trabalho não envolveu o desenvolvimento e implementação direta da arquitetura aqui apresentada. Os fundamentos que preparam o sistema para o uso de *Event Sourcing*, CQRS, e DDD já estavam disponíveis, e sua construção é responsabilidade de desenvolvedores mais experientes da equipe. A compreensão detalhada dos padrões foi, no entanto, necessária para a implementação de novas funcionalidades da API, assim como para a manutenção de código já existente. Por isso, nesta sessão será também descrito como se dá o desenvolvimento baseado nessa arquitetura no sistema da Doare, com detalhamento das etapas da construção de novas funcionalidades que foram responsabilidades de fato exercidas no escopo deste trabalho.

4.5.1 *Event Sourcing* e CQRS - Event Store, Projeções e banco de dados MySQL/MySQL

O sistema da Doare utiliza a ferramenta *Event Store* para armazenar as informações relacionadas ao sistema de Doações utilizando a ideia de *Event Sourcing*. Para a escrita de novas informações no sistema, tudo se dá em termos de eventos. Antes um evento relacionado a uma instância de uma entidade ser adicionado, todos os eventos relacionados a esse objeto são resgatados da *Event Store*. Cada um desses eventos causa algum tipo de mudança nas características do objeto, que são definidas como funções de aplicação dos eventos. Um objeto é então criado e tem suas características alteradas sequencialmente para cada um dos eventos resgatados.

O objeto recriado a partir dos eventos é então processado e recebe, conforme o contexto, novos eventos. O novo conjunto de eventos é então salvo quando o processamento é concluído.

Passar por este processo aparentemente complicado no momento da escrita de novos eventos é um revés necessário em *Event Sourcing*. A informação presente no armazenamento

de eventos é a fonte de verdade, e ela nunca pode ser corrompida. É, portanto, importante que novas operações levem eventos antigos em consideração, e que os eventos sejam sempre salvos de maneira atômica, sem que possa haver inconsistências em sua sequência. No entanto, para ações de leitura, que, como dito anteriormente, costumam ser mais requisitadas, toda essa complexidade não é necessária e pode mostrar-se lenta demais. Essa diferença de requisitos entre leitura e escrita de dados no sistema motiva a implementação do conceito de CQRS no sistema.

Para garantir que a escrita de eventos seja consistente enquanto a leitura de dados seja rápida, é prática comum em sistemas com *Event Sourcing* manter um banco de dados relacional onde fica armazenado o estado atual do sistema. Esse banco de dados é atualizado sempre que um novo evento é criado, por meio de projeções, que escutam os eventos e realizam as alterações definidas pelos eventos também no banco de dados relacional. Quando ocorre, portanto, uma ação de leitura, ou consulta, a informação pode ser recuperada diretamente do banco de dados, sem que seja necessário que um objeto seja reconstruído a partir de seus eventos.

No sistema da Doare essa separação entre consultas e comandos é implementada. O banco de dados relacional utilizado é o MariaDB, uma implementação alternativa de MySQL. Trata-se de um banco de dados largamente utilizado na industria. A linguagem utilizada para leitura e escrita nesse banco de dados é o SQL, assim como na maioria dos bancos de dados relacionais mais utilizados. Para a comunicação entre o software desenvolvido em *TypeScript* e o banco de dados MariaDB é utilizado o pacote TypeScript *Store* (*@cashfarm/store, no gerenciador de pacotes npm*), um framework de mapeamento objeto-relacional (ORM) de código aberto desenvolvido por um dos membros da Doare.

4.5.2 Implementação de nova funcionalidade no sistema

Para que todos esses conceitos sejam aplicados de maneira sistemática no desenvolvimento da API da Doare, toda a sua fundamentação é realizada da maneira mais genérica possível, de forma que os desenvolvedores que criam novas funcionalidades só precisam compreender os conceios e desenvolver utilizando a estrutura criada como um framework sobre o qual trabalhar. Boa parte dessa infraestrutura está disponível em pacotes de código aberto como o já citado *Store*, *Plow (@cashfarm/plow)*, entre outros do mesmo desenvolvedor pertencente ao time. Outra parte desses fundamentos está na preparação do projeto para integração com esses pacotes, deixando o sistema pronto para a implementação de novas funcionalidades que se beneficiam do DDD, do *Event Sourcing*, e do CQRS. O processo de desenvolvimento passa então a ser sistemático e semelhante para novas funcionalidades criadas, e pode ser descrito por algumas etapas:

- Se a entidade com a qual se está trabalhando ainda não existe, ela deve criada com

seus atributos e eventos. Para uma nova entidade criada, também é necessário criar um repositório, que é um objeto de interface com a *Event Store*. Se a entidade já existe, muitas vezes só é necessário modelar um novo evento. Em alguns casos ainda o evento necessário já existe. Para novos eventos criado, é necessário criar os métodos de aplicação desses eventos, para determinar o que ocorre com um objeto quando são aplicados.

- A criação ou alteração de um controlador (*Controller*), que serve como interface da API, determinando as requisições REST que podem ser recebidas pelo sistema. Requisições da API chegam a um dos terminais (*Endpoints*) do controlador, são processadas por ele, que, se for o caso, aciona os eventos necessários e envia uma resposta. Controladores são desenvolvidos no sistema da Doare separadamente para cada entidade modelada no sistema. Há um controlador para transações, outro para organizações e outro para campanhas, por exemplo.

- Também deve ser criada, se ainda não existir, a ligação com o modelo de leitura (*readModel*) do sistema para os objetos com os quais se está trabalhando . Cria-se, portanto, um objeto do tipo *Store*, que utiliza a biblioteca *Store*, já citada, e fornece uma interface com o banco de dados MariaDB que deve conter o estado atual do sistema. O objeto *Store* (*TransactionStore, CampaignStore*, por exemplo), oferece métodos de leitura e escrita de dados de uma tabela do banco de dados.

- Devem ser criadas também projeções para os eventos criados. Essas projeções escutam determinados eventos e realizam as alterações adequadas no banco de dados quando algum desses eventos ocorre. Com isso o banco de dados se mantém consistente com a informação presente na *Event Store*.

5 Participação direta no desenvolvimento do projeto

Como já foi exposto anteriormente, este trabalho consistiu na colaboração com os trabalhos de um time de desenvolvedores na construção do sistema de doações da Doare. Por ser um sistema grande e complexo, para que se pudesse contribuir com sua construção foi necessária uma compreensão de todo o seu funcionamento. Com esse conhecimento em mãos foi possível o desenvolvimento e manutenção de algumas partes importantes do Software. Este capítulo apresenta brevemente as principais contribuições realizadas no período de duração deste trabalho para o sistema.

5.1 Integração com meios e *gateways* de pagamento

Como já foi comentado anteriormente neste documento, uma característica particular do sistema da Doare é que ele é capaz de receber pagamentos de diversas maneiras diferentes. Um usuário pode escolher diversos meios de pagamento, como cartão de crédito, ou boleto bancário, para realizar seus pagamentos, e o sistema, por sua vez, pode processar cada pagamento através de diferentes *gateways* de pagamento, como Moip, Stripe, ou Paypal. Isso propicia uma flexibilidade que permite que os pagamentos sejam realizados da melhor maneira possível a cada momento, ou seja, onde houver as menores taxas e melhores serviços.

O desenvolvimento de um software capaz de adaptar-se a tantas situações diferentes torna-se, no entanto, bastante complexo. Cada *gateway* de pagamento fornece uma API completamente diferente com a qual o sistema deve se comunicar, assim como cada meio de pagamento demanda uma lógica diferente da aplicação para ser processado. Alguns problemas podem surgir dessa complexidade, sendo que o maior deles é uma possível falta de consistência entre as informações de pagamento disponíveis no próprio *gateway*, e as informações relacionadas no sistema da Doare. Essas duas fontes de dados precisam estar sempre sincronizadas, já que se trata de operações financeiras e, portanto, críticas. Em algumas situações isso não ocorre, o que pode ser causado por um erro de programação, por uma pane no sistema que deixa ele fora do ar por algumas horas, ou por outras razões imprevisíveis. Uma transação paga pode constar como não paga no sistema, ou um estorno pode não ter sido devidamente computado, por exemplo. Uma quantia considerável de trabalho de manutenção é necessária, portanto, para prevenir e remediar esse tipo de situação.

5.1.1 *Webhooks* - Notificações dos *Gateways* de pagamento

Uma das maneiras pela qual se dá a conexão entre os *gateways* de pagamento e o sistema da Doare é através de *endpoints* especiais da API da Doare chamados *webhooks*. Um *webhook* é um sistema de notificação que um sistema utiliza para avisar outro sistema de algo. O *gateway* Moip, por exemplo, utiliza um *webhook* para notificar o sistema da Doare de que um boleto de doação foi pago ou de que seu prazo de pagamento expirou. O desenvolvimento e manutenção de alguns desses *webhooks* foi uma parte significativa do trabalho que origina este documento. Um *webhook* é um *endpoint* como qualquer outro no sistema. Cadastra-se ele no *gateway* de pagamento conforme orientado pela documentação do mesmo. A API da Doare passa então a receber requisições REST que trazem as informações do evento que acaba de ocorrer (o pagamento de um boleto, por exemplo). O *controlador* que possuí esse *endpoint* trata então a informação, acionando os Eventos adequados dentro do sistema. No caso do pagamento de um boleto, o controlador buscaria a instância da entidade transação a qual ele se refere, e acionaria o evento transação paga dessa instância. Esse evento seria tratado como qualquer outro dentro do sistema, tendo suas informações persistidas na *Event Store* e sendo projetado no banco de dados de leitura, como foi esclarecido no capítulo anterior.

5.1.2 *Scripts* que restabelecem consistência de dados

Em situações nas quais as informações de pagamento já estão corrompidas, é necessário garantir que sua consistência seja reestabelecida. Para isso torna-se necessária a programação de *scripts* que uma vez executados fornecem uma maior certeza de que os dados presentes no sistema estão corretamente sincronizados com os *gateways* de pagamento. Este *script* pode então ser executado periodicamente no sistema de maneira automática ou conforme necessidade. No escopo deste trabalho, a principal contribuição relacionada a isso foi a criação de um *script* que atualiza todas as transações de boleto bancário do *gateway* Moip para garantir que seu *status* no sistema está correto. O *script* desenvolvido utiliza a API fornecida pelo Moip para obter informações sobre os milhares de boletos cadastrados pela Doare no *gateway*. Cada um desses boletos é buscado, utilizando-se uma referência, no sistema, para então verificar-se se os *status* nos dois sistemas coincidem. Caso exista uma divergência ela é então corrigida. Para realizar essa correção o *script* simula uma requisição REST idêntica à que o Moip realizaria para notificar a alteração de *status* referente.

5.2 Integração entre o sistema legado e a nova API

Todas as novas funcionalidades do sistema da Doare vêm sendo implementadas no novo sistema desenvolvido em *Typescript/NodeJs*. No entanto, há um sistema antigo em

PHP, com um banco de dados próprio que deve ser mantido atualizado junto ao banco de dados novo. Esse sistema ainda é responsável por boa parte das operações realizadas no sistema, ainda não foi possível substitui-lo. Para que esta sincronia entre os bancos de dados seja mantida, deve ser realizada a projeção de toda informação nova no sistema novo para o sistema antigo. Uma das maneiras de realizar isso é bastante direta - assim como há projeções que escutam eventos e escrevem informações no banco de dados de leitura do sistema novo, outras projeções podem escutar os mesmos eventos e atualizar o banco de dados do sistema legado. Algumas vezes, no entanto, isso não foi realizado a tempo e o sistema legado fica desatualizado em relação ao sistema novo. Nesse caso, *scripts* semelhantes ao que acaba de ser descrito para os boletos foram criados, ou seja, *scripts* que compararam as transações do sistema novo e do sistema legado e atualização as informações incorretas.

5.3 Implementação de novas funcionalidades e Ajustes na interface gráfica do sistema

No contexto deste trabalho também foram realizadas diversas mudanças na interface gráfica da plataforma de doações da Doare. Essas mudanças normalmente envolviam a criação de novos *endpoints* na API para obtenção de informações do sistema ou para o envio de algum comando para o sistema. Com a API preparada para a nova funcionalidade tornava-se possível a implementação das mudanças nas páginas da plataforma de doações ou até a implementação de novas páginas.

6 Conclusões e Perspectivas

O desenvolvimento de aplicações para a internet é uma das atividades mais intensas atualmente na industria de software. Este trabalho e as atividades que o envolveram serviram como um excelente ponto de partida e de aquisição de experiências nessa área. As ferramentas de desenvolvimento utilizadas no desenvolvimento do sistema, uma rápida pesquisa na industria comprova, são recursos utilizados largamente e cujo conhecimento será de grande valia. Durante o trabalho fez-se contato com padrões de arquitetura de software consideravelmente complexos, modernos *frameworks* em *javascript* e *typescript* para o desenvolvimento de aplicações back-end e front-end, assim como técnologias de infraestrutura de internet como servidores e diferentes tipos de bancos de dados.

A participação no time de desenvolvedores de uma empresa de tecnologia ainda pequena propiciou uma oportunidade técnica valiosa de entrar em contato com boa parte do leque de tecnologias que possibilitam o projeto, construção e manutenção de um sistema da internet, assim como uma oportunidade profissional e pessoal única de contribuir com um produto que passa um processo acelerado de construção, uso e mudança.

Ainda mais importante do que isso, já que a engenharia só vale de fato o quanto ela contribuí para a sociedade como um todo, o serviço fornecido pelo sistema abordado neste trabalho tem uma ligação direta com inúmeras instituições que lutam por diferentes causas no Brasil e no mundo. Trabalhar em algo assim motivador é também essencial para o bem estar pessoal, e é de grande ajuda para alcançar um maior engajamento com as atividades do dia a dia. Contribuir com este projeto foi uma experiência que demonstrou isso com clareza, e que fará com que outras atividades tão significativas quanto essa sejam buscadas no futuro.

A construção deste documento foi importante no fim da curta duração desde estágio para que os conhecimentos adquiridos nele se consolidassem e que para que se tomasse consciência de como o projeto se conecta com o curso de Engenharia de Controle e Automação. De fato, os conhecimentos e experiências trazidos pelo curso foram de grande ajuda no dia a dia de desenvolvimento. De maneira mais evidente, foram úteis os conteúdos obtidos em toda a parte do curso que se relaciona à Tecnologia da Informação de alguma forma. As matérias Introdução à Informática para Automação, Fundamentos da Estrutura da Informação, Sistemas Digitais, Arquitetura e Programação de Sistemas Microcontrolados, Metodologia para desenvolvimento de Sistemas e Programação Concorrente e Sistemas de Tempo Real foram as que mais diretamente contribuíram como base para o desenvolvimento de software realizado na empresa, assim como na escrita da monografia. O curso como um todo contribuiu para o desenvolvimento do pensamento analítico e abstrato necessário

para a compreensão detalhada da arquitetura do sistema da empresa e desenvolvimento de algoritmos de maior qualidade.

O plataforma de Doações da Doare é um produto ainda em processo de amadurecimento e que passa por mudanças frequentes, tanto na forma como ele é oferecida para seus usuários finais, quanto no software que o forma. Num futuro próximo, na continuação de esforços que ocorreram próximo ao termino do estágio que origina este trabalho, o painel no qual o organizador de uma ONG pode acompanhar o andamento de suas campanhas será totalmente reconstruído com Angular e utilizará somente o novo sistema. Objetiva-se com isso diminuir cada vez mais a dependência do sistema legado da empresa. Além disso, a Doare passará em breve a contar com um novo aplicativo de Doações pelo celular, que será mantido pela equipe técnica da empresa.

Pessoalmente, este trabalho firmou a motivação de seguir na área de desenvolvimento de software, particularmente ná área de desenvolvimento da infraestrutura de sistemas *web*, ou *Back-end*.

Referências

1 MICROSOFT. - introducing the command query responsibility segregation pattern - disponível em «https://msdn.microsoft.com/en-us/library/jj591573.aspx». Citado 3 vezes nas páginas 9, 47 e 48.

2 GUEDES, G. *UML 2 - Uma Abordagem Prática*. [S.l.]: Novatec, 2011. Citado na página 14.

3 B, M. C. C. Scrum agile product development method - literature review, analysis and classification. 2011. Citado na página 15.

4 MULESOFT. - what is an api - disponível em «https://www.mulesoft.com/resources/api/what-is-an-api». Citado na página 29.

5 - Documentação Angular - disponível em «https://angular.io/docs». Citado 2 vezes nas páginas 31 e 32.

6 HUSTON, T. What is microservices architecture. Citado na página 37.

7 THIJSSEN, J. The restful cookbok - disponível em «http://restcookbook.com/». Citado na página 39.

8 EVANS, E. *Domain Driven Design*. [S.l.]: Alta Books, 2016. Citado 2 vezes nas páginas 39 e 40.

9 - DDD, Event Sourcing and CQRS Tutorial - Disponível em «http://cqrs.nu/tutorial/cs/01-design». Citado na página 46.

10 MEYER, B. *Object Oriented Software Contstruction*. [S.l.: s.n.], 1988. Citado na página 46.

11 MICROSOFT. - introducing event sourcing - disponível em «https://msdn.microsoft.com/en-us/library/jj591559.aspx». Citado na página 48.