



Universidade Federal do Rio de Janeiro
Escola Politécnica / COPPE

Doador Sangue Bom – Um sistema para promoção de doações de sangue

Autor:

Renan Bernardo Valadão

Autor:

Thiago Xavier Cardoso

Orientador:

Prof. Sergio Barbosa Villas-Boas, Ph.D.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Examinador:

Robson Pereira de Lima, Dr.

Poli / COPPE
Setembro de 2012

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica – Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária.

Rio de Janeiro – RJ – CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmар ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do autor e do orientador.

Valadão, Renan Bernardo

Cardoso, Thiago Xavier

Doador Sangue Bom – Um Sistema de Promoção de Doações de Sangue / Renan Bernardo Valadão e Thiago Xavier Cardoso – Rio de Janeiro: UFRJ/POLI-COPPE, 2012.

XII, 70 p.: il.; 29,7 cm.

Orientador: Sergio Barbosa Villas-Boas

Projeto (Graduação) – UFRJ/POLI/Departamento de Eletrônica e de Computação – COPPE, 2012.

Referências Bibliográficas: p. 71-75.

1. Serviços Web 2. Doação de Sangue 3. Arquitetura Orientada a Serviços 4. Inovação Social I. Villas-Boas, Sergio Barbosa II. Universidade Federal do Rio de Janeiro, Poli/COPPE. III. Título.

AGRADECIMENTO

Renan Bernardo Valadão

À minha mãe, Mariza Bernardo, e à minha irmã, Talita Bernardo.

Ao parceiro de projeto, Thiago Xavier.

Aos amigos Bárbara Danin, Barbara Jardim, Camila Rebello, Carlos Pivotto, Diniz Viegas, Eduardo Moura, Fábio Fonseca, Fernanda Vinhas, Gustavo Fernandes, Hugo Eiji, João Luiz Ferreira, Jorge Valdivia, João Pedro Francese, Juliana Bunn, Juliana Fayad, Laís Boffy, Pedro Henrique Nunes, Pedro Pisa, Rafael Baur, Rafael Studart, Raquel Torres, Roberta Costa, Roosevelt Sardinha, Sérgio Quezado, Silvia Benza, Thiago Nascimento, Thiago Neves, Tiago Miquelino, Túlio Ligneul, Ubirajara Machado e Victor Mendes.

Ao professor Sergio Barbosa Villas-Boas, por sua orientação e participação essencial na formação de um Engenheiro de Computação e Informação.

Ao companheiro João Purificação, por sua orientação na parte de DOUA.

Ao povo brasileiro, por contribuir para minha formação superior.

Ao coordenador no curso, José Ferreira de Rezende, por sua dedicação aos alunos.

Thiago Xavier Cardoso

À minha mãe, Silvana Xavier, e ao meu pai, Virgílio Cardoso.

Ao parceiro de projeto, Renan Bernardo.

Aos amigos Bruno Rosa, Carlos Pivotto, Diniz Viegas, Fábio Fonseca, Felipe Alves, Felipe Mello, Fernanda Vinhas, Guilherme Martins, Gustavo Fernandes, Hugo Eiji, João Luiz Ferreira, João Pedro Francese, Jorge Valdivia, Juliana Bunn, Juliana Fayad, Laís Boffy, Maiara Oliveira, Natasha Paes Leme, Nathália Portugal, Pedro Henrique Nunes, Pedro Pisa, Rafael Baur, Rafael Studart, Raisal Lima, Raquel Torres, Roberta Costa, Roberta Russo, Roosevelt Sardinha, Silvia Benza, Túlio Ligneul e Ubirajara Machado.

À amiga Camila Rebello, médica, por seu interesse e contribuições técnicas da medicina para o projeto.

À amiga Luciana Hilgenberg, administradora, por contribuir com o nome do projeto.

Ao amigo Yuri Magno, designer, por contribuir com o logo do sistema.

Ao professor Sergio Barbosa Villas-Boas, por sua orientação, motivação e participação essencial na formação de um Engenheiro de Computação e Informação.

Ao companheiro João Purificação, por sua orientação na parte de DOUA.

Ao povo brasileiro, por contribuir para minha formação superior.

Ao coordenador no curso, José Ferreira de Rezende, por sua dedicação aos alunos.

RESUMO

Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

DOADOR SANGUE BOM – UM SISTEMA DE PROMOÇÃO DE DOAÇÕES DE SANGUE

Renan Bernardo Valadão
Thiago Xavier Cardoso

Setembro/2012

Orientador: Sergio Barbosa Villas-Boas
Curso: Engenharia de Computação e Informação

Visando promover doações de sangue entre usuários da *Internet*, foi desenvolvido um sistema por meio da linguagem de programação Java. Foram construídos um *web service*, um aplicativo *web* e um aplicativo para dispositivos móveis com o sistema operacional Android. Como a divulgação e incentivo às doações de sangue no Brasil são muito tímidas, o sistema representa um tipo de inovação social que contribui para a área da saúde pública.

Palavras-Chave: Serviços Web, Doação de Sangue, Arquitetura Orientada a Serviços, Inovação Social.

ABSTRACT

Undergraduate Project presented to POLI/COPPE/UFRJ as a partial fulfillment of requirements for the degree of Computer and Information Engineer.

DOADOR SANGUE BOM – A SYSTEM TO PROMOTE BLOOD DONATIONS

Renan Bernardo Valadão

Thiago Xavier Cardoso

September/2012

Advisor: Sergio Barbosa Villas-Boas

Course: Computer and Information Engineering

Aiming to promote blood donations among Internet users, a system was developed using the Java programming language. We constructed a web service, a web application and an application for mobile devices with the Android operating system. As the dissemination and encouragement of blood donations in Brazil are very shy, the system represents a kind of social innovation that contributes to public health.

Keywords: Web Services, Blood Donations, Service-Oriented Architecture, Social Innovation.

SIGLAS

A-GPS *Assisted Global Positioning System*
ADT *Android Development Tools*
API *Application Programming Interface*
APK *Android Application Package*
CNES *Cadastro Nacional de Estabelecimentos de Saúde*
CSV *Comma-separated values*
DAO *Data Access Object*
DDMS *Dalvik Debug Monitor Server*
DOUA *Database Oriented Usecase Authorization*
EDGE *Enhanced Data rates for GSM Evolution*
FTP *File Transfer Protocol*
GPS *Global Positioning System*
HTTP *HyperText Transfer Protocol*
IDE *Integrated Development Environment*
IP *Internet Protocol*
JDBC *Java DataBase Connectivity*
JSF *JavaServer Faces*
JSON *JavaScript Object Notation*
OMS *Organização Mundial da Saúde*
REST *REpresentational State Transfer*
SGBD *Sistema de Gerenciamento de Banco de Dados*
SQL *Structured Query Language*
SMTP *Simple Mail Transfer Protocol*
SOA *Service-Oriented Architecture*
SOA-MC *Service-Oriented Architecture - Multiple Clients*
SOAP *Simple Object Access Protocol*
SVN *Apache Subversion*
UDDI *Universal Description Discovery and Integration*
URI *Uniform Resource Identifier*
URL *Uniform Resource Locator*
W3C *World Wide Web Consortium*
WSDL *Web Services Description Language*
XHTML *eXtensible Hypertext Markup Language*
XML *eXtensible Markup Language*

Sumário

Capítulo 1 - Introdução	1
1.1 Motivação	1
1.2 Objetivo.....	4
1.3 Metodologia	4
1.4. Estrutura do Documento	5
Capítulo 2 - O Doador Sangue Bom	7
2.1. Como Funciona.....	7
2.2. Inovação Social.....	9
Capítulo 3 - O banco de dados do Doador Sangue Bom	12
3.1. A escolha do SGBD	12
3.2. Modelagem	13
3.3. DOUA - <i>Database Oriented Usecase Authorization</i>	16
Capítulo 4 - <i>Web Services</i> e SOA-MC - Teoria	18
4.1. Arquitetura Orientada a Serviços - Múltiplos Clientes.....	18
4.2. <i>Web Services</i>	20
Capítulo 5 - O <i>Web Service</i> do Doador Sangue Bom.....	28
5.1. Tecnologias utilizadas.....	28
5.2. Criação do <i>Web Service</i>	29
5.3. Os métodos do Doador Sangue Bom.....	31
Capítulo 6 - Cliente I - Aplicação <i>Web</i>	35
6.1. Tecnologias utilizadas.....	35
6.2. <i>JavaServer Faces</i> (JSF).....	36
6.2.1. Ciclo de Vida	37
6.2.1.1. Restore View	37
6.2.1.2. Apply Request Values	38
6.2.1.3. Process Validation.....	38
6.2.1.4. Update Model Values	38
6.2.1.5. Invoke Application	38
6.2.1.6. Render Response.....	39
6.2.1.7. Process Events	39
6.3. Internacionalização do Sistema.....	39
6.4. O <i>Website</i> do Doador Sangue Bom	40
6.4.1. Interface Doador/Receptor	41
6.4.1.1. Cadastro de Usuário (cadastroUsuario.xhtml)	41
6.4.1.2. Página Principal (doarSangue.xhtml).....	41
6.4.1.3. Doe (doe.xhtml)	42
6.4.1.4. Receba (requisitarDoacao.xhtml)	43
6.4.1.5. Alterar Cadastro (editarUsuario.xhtml).....	43
6.4.1.6. Histórico de Doações (historicoDoacoes.xhtml).....	44
6.4.1.7. Hospitais (hospitais.xhtml).....	44
6.4.1.8. Informações (informacao.xhtml)	45

6.4.1.9. Visualizar Doação (visualizarDoacao.xhtmll)	45
6.4.2. Interface Coletor	46
6.4.2.1. Cadastro de Coletor (cadastroColetor.xhtmll)	46
6.4.2.2. Validar Doações (validarDoacao.xhtmll)	47
6.4.3. Interface Administrador	47
6.4.3.1. Administração de Usuários (adminUsuario.xhtmll)	47
6.4.3.2. Administração de Hospitais (adminHospital.xhtmll)	48
6.4.3.3. Notícias (cadastroNoticia.xhtmll)	49
6.4.3.4. Gerar Código de Coletor (adminCodigoColetor.xhtmll)	49
6.4.3.5. Estatísticas (estatisticas.xhtmll)	50
Capítulo 7 - Cliente II - Aplicativo Android	51
7.1. A escolha do sistema operacional móvel	52
7.2. Tecnologias utilizadas	53
7.3. Considerações técnicas	53
7.3.1. Geolocalização	54
7.3.2. <i>ASyncTask</i>	55
7.4. O aplicativo do Doador Sangue Bom	57
7.4.1. <i>Login</i> e tela inicial	57
7.4.2. Visualização de Doações	58
7.4.3. Doação específica e mapa	59
7.4.4. Requisitar doação	60
7.4.5. Visualização de hospitais	60
7.4.6. Notícias	61
7.4.7. Cadastro e edição de usuário	61
7.4.8. Histórico	62
7.5. Publicação no <i>Google Play</i>	63
Capítulo 8 – Conclusões	67
Capítulo 9 - Trabalhos Futuros	68
9.1. Aplicativo para iOS	68
9.2. Funcionalidades Além da Doação de Sangue	68
9.3. Modelo de Negócios	68
9.4. Nível de Urgência para Doações	69
9.5. Hibernate	69
9.6 Estatísticas Regionais de Doações	69
Referências	71

Lista de Figuras

Figura 1.1. Frequência de doação de sangue	2
Figura 1.2. Disponibilidade para doação em relação à localização	2
Figura 1.3. Destino das doações de sangue	3
Figura 1.4. Requisição de doações de sangue	3
Figura 1.5. Retorno das doações de sangue	3
Figura 2.1. Processo de validação de doações de sangue	8
Figura 2.2. Índice de doadores em relação à população	10
Figura 2.3. Total de bolsas de sangue coletadas pela hemorrede pública	10
Figura 3.1. Modelo do banco de dados	14
Figura 3.2. Modelo do DOUA	17
Figura 4.1. Diretório de serviços	19
Figura 4.2. SOA-MC	20
Figura 4.3. Mensagem SOAP	22
Figura 4.4. Esqueleto de um WSDL	23
Figura 4.5. WSDL	23
Figura 4.6. Troca de mensagens SOAP	24
Figura 4.7. Requisições em um serviço <i>RESTful</i>	26
Figura 5.1. Senha em hash	30
Figura 5.2. <i>Web Service</i> e clientes	31
Figura 6.1. Separação de camadas no aplicativo web	37
Figura 6.2. Ciclo de Vida JSF	37
Figura 6.3. Idiomas do Doador Sangue Bom	40
Figura 6.4. Página principal	42
Figura 6.5. Mapa de todos os locais de doação	43
Figura 6.6. Página de visualizar doação	46
Figura 6.7. Validação de doações	47
Figura 6.8. Página de administração de hospitais	49

Figura 7.1. Tela de login do aplicativo.....	58
Figura 7.2. Tela de visualização de doações do aplicativo.....	59
Figura 7.3. Tela de mapa do aplicativo	60
Figura 7.4. Tela de edição de usuário do aplicativo	61
Figura 7.5. Tela de inserção de doação em histórico no aplicativo	62
Figura 7.6. AndroidManifest	64
Figura 7.7. Publicação no Google Play	65
Figura 7.8. Publicação no Google Play II	66

Lista de Tabelas

Tabela 5.1. Ferramentas utilizadas para desenvolvimento do <i>Web Service</i>	29
Tabela 5.2. Métodos do Doador Sangue Bom	34
Tabela 6.1. Ferramentas utilizadas para desenvolvimento da aplicação <i>web</i>	36
Tabela 7.1. Ferramentas utilizadas para desenvolvimento do aplicativo Android.....	53

Capítulo 1 - Introdução

1.1 Motivação

A promoção de doações de sangue no Brasil nunca foi largamente divulgada. Apesar da quantidade de indivíduos necessitados de doações ser enorme, seu alcance é muito baixo e os meios de comunicação utilizados não são aconselháveis para pedidos de doação de sangue, deixando hospitais com poucas bolsas de sangue e pacientes em risco de vida.

Após a ascensão da era da informação e da Internet, pedidos de doação de sangue começaram a ser feitos via e-mail e, mais recentemente, por meio de redes sociais como o Facebook ou Twitter. Apesar da praticidade e do alcance que esses meios podem apresentar, o que acontece na prática é que poucos pedidos são realmente atendidos.

Para este trabalho, realizamos uma pesquisa com cerca de 480 pessoas através de um formulário divulgado em redes sociais e nos grupos de e-mail da Escola Politécnica [1]. A pesquisa contou com as seguintes perguntas:

1. Com qual frequência você costuma doar sangue?
2. Se o local de doação fosse perto de sua casa ou trabalho você se sentiria mais incentivado(a) a doar sangue para um anônimo?
3. Caso já tenha doado, você já atendeu a um pedido de doação que lhe foi encaminhado por e-mail ou rede social?
4. Se a resposta à pergunta anterior foi "Sim", para quem foi direcionada essa doação?
5. Você já requisitou uma doação de sangue por e-mail ou rede social?
6. Se a resposta à pergunta anterior foi "Sim", qual foi o retorno desse pedido?

Para a primeira questão, como visto na Figura 1.1 foram obtidos os seguintes resultados: 52% disseram que nunca doaram, 17% disseram que doam uma ou duas vezes por ano, 17% disseram que não têm o hábito de doar, mas já doaram, 8% afirmaram não poder doar sangue, 3% disseram doaram de três a quatro vezes por ano e 3% disseram que doam apenas quando uma pessoa próxima precisa. Assim, podemos constatar que o número de pessoas que nunca doaram ou não possuem o hábito de doar é muito grande. Tendo isso em vista, este trabalho propõe a criação de

um sistema que busca incentivar doações por parte dessas pessoas. Esse incentivo será proporcionado indicando pessoas que necessitam de doação em locais próximos ao endereço do usuário.

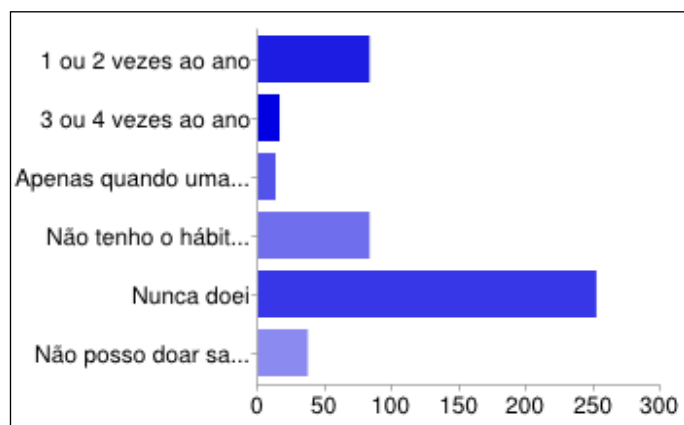


Figura 1.1. Frequência de doação de sangue

A Figura 1.2, relativa à segunda pergunta, mostra que 80% dos participantes da pesquisa se sentiriam incentivados a doar sangue se o local de doação fosse próximo ao seu endereço.

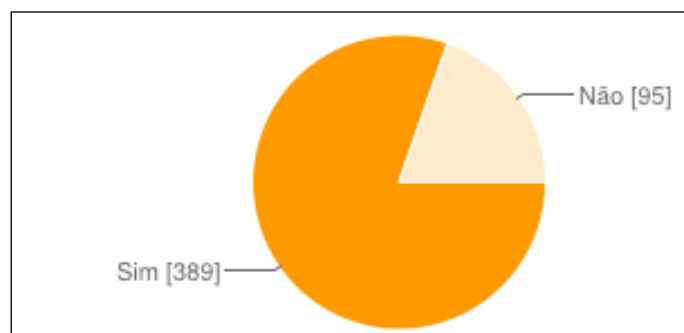


Figura 1.2. Disponibilidade para doação em relação à localização

Para a terceira pergunta, a maioria dos participantes afirmou que nunca atendeu um pedido de doação feito por e-mail ou rede social. O sistema desenvolvido para este trabalho visa também atrair pessoas que tenham o hábito de doar, facilitando a visualização de doações em um ambiente limpo e de simples visualização.

Como mostra a Figura 1.3, a maior parte das pessoas que atendeu a um pedido de doação encaminhado por e-mail ou rede social, o fez para uma campanha de doação de sangue. Outra ideia deste trabalho é que o número de doações para

pacientes individuais também aumente, fazendo com que as pessoas não sejam incentivadas apenas quando houver campanhas.

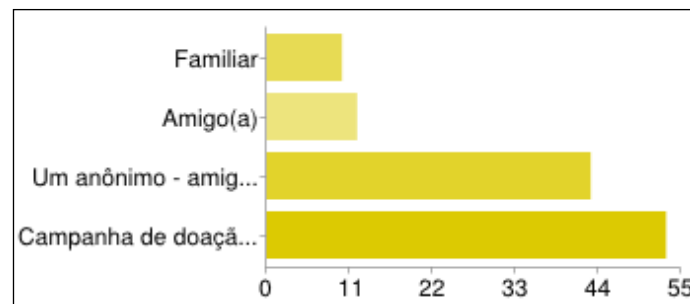


Figura 1.3. Destino das doações de sangue

De acordo com as Figuras 1.4 e 1.5, relativas à quinta e sexta perguntas, algumas pessoas já requisitaram doações através de e-mail ou rede social, porém, obtiveram um retorno muito baixo ou nulo. Isso demonstra que pedidos de doação feitos apenas através desses meios de comunicação são ineficientes e necessitam de um ambiente complementar onde as solicitações de doação tenham um aspecto mais confiável.

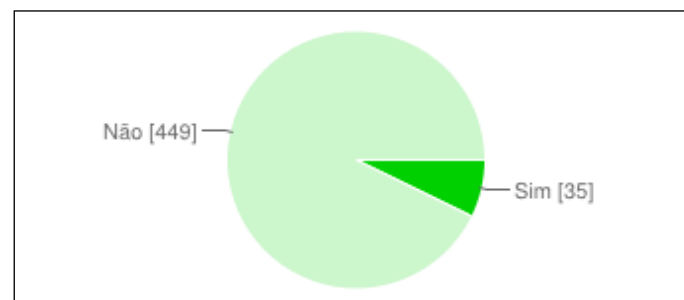


Figura 1.4. Requisição de doações de sangue

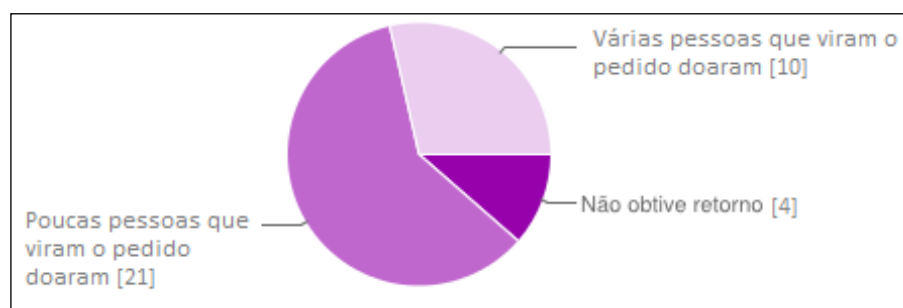


Figura 1.5. Retorno das doações de sangue

O ato de doar sangue requer altruísmo e tempo, duas características que podem ser difíceis de conciliar, ainda mais com pedidos feitos de forma informal em redes sociais e e-mail, muitas vezes tornando-se difícil ou impossível de verificar se o pedido é verídico ou se é algum tipo de fraude ou informação falsa. Endereços

errados, distantes ou não disponíveis, além de outras informações errôneas, tornam ainda mais difícil e desorganizado o processo de requisição e de doação de sangue utilizando os meios de comunicação supracitados.

Com esse cenário foi visualizada a possibilidade de desenvolvimento de um sistema para gerenciar pedidos e promover doações de sangue, atacando diversos pontos fracos que os pedidos de doação via Internet apresentam atualmente e apresentando funcionalidades para facilitar tanto o processo de solicitação de uma nova doação, como a organização das doações de um determinado hospital.

1.2 Objetivo

O objetivo deste trabalho é projetar e implementar um projeto de *software* que consiste em um *Web Service* e duas aplicações clientes baseadas em SOA-MC, uma arquitetura orientada a serviços utilizada por múltiplos clientes, que será detalhada mais adiante. Uma das aplicações clientes será para a *web*, acessada via navegador, e outra para *smartphones* baseados no sistema operacional Android, do Google. Ambas as aplicações consumirão o *Web Service* desenvolvido. Essas aplicações têm o objetivo de prover uma solução para os problemas explicados na seção anterior, facilitando o processo de doação de sangue e motivando os doadores.

O *Web Service* se conectará com um banco de dados remoto, também modelado pelos membros do projeto, onde serão armazenados os dados dos usuários do sistema, que poderão ser acessados em ambas aplicações clientes.

No aplicativo *web*, os hospitais também poderão ter controle sobre as doações intermediadas pelo sistema, através da figura do coletor.

O aplicativo Android será publicado no Google Play (antigo Android Market) [2], a loja *online* de aplicativos do Google. No entanto, como o sistema não estará em produção ao término do projeto, a publicação possuirá simplesmente o objetivo de aprendizado.

O nome escolhido para tal sistema foi Doador Sangue Bom.

1.3 Metodologia

Dentre as diversas metodologias de desenvolvimento de software estudadas na graduação, em disciplinas como Engenharia de Software ou Linguagens de Programação, a escolhida para este trabalho foi o *Scrum Academic*. Essa metodologia consiste de fazer versões iterativas e incrementais do *software*, cujo código fonte é enviado para um repositório.

O *Scrum Academic* requer várias reuniões com o usuário, que avaliará e discutirá possíveis mudanças ou melhorias no projeto, além de definir prioridades. Para este trabalho, o usuário final foi representado pelo orientador e a “equipe *Scrum*” são os dois membros do projeto. Estes devem apresentar versões cada vez mais próximas da versão final a cada reunião, que serão devidamente avaliadas pelo usuário final. Essas reuniões ocorrerão até a conclusão do projeto.

O *Scrum Academic* é uma vertente do método *Scrum* e difere deste pelo fato de ser aplicado a projetos acadêmicos e possuir um escopo menor e com menos foco comercial.

Os dois integrantes do projeto utilizaram também o Trello para organizar ideias e definir o que está sendo realizado, além do que está pendente. O Trello é uma ferramenta desenvolvida para *web* e *smartphone*, para organização e gerência de ideias ou tarefas.

O projeto foi versionado utilizando-se o SVN e a ferramenta Tortoise SVN [3] para baixar e enviar as versões atualizadas. O repositório está acessível tanto para o orientador como para os integrantes do projeto no endereço: <http://selectos.indefero.net/svn/selectos/doarsangue>.

Por fim, a modelagem do sistema seguiu o padrão de orientação a objetos, com o objetivo de facilitar o entendimento do problema e obter uma maior organização de todas as partes do projeto. Além disso, a orientação a objetos é uma forma eficiente de implementar o sistema, e que torna mais fácil a manutenção do mesmo.

1.4. Estrutura do Documento

O capítulo 2 explica como funciona e quais as utilidades do Doador Sangue Bom e por que pode ser caracterizado como uma inovação social.

O capítulo 3 aborda a modelagem do banco de dados do Doador Sangue Bom e explica cada uma de suas tabelas. Além disso, há uma justificativa para a escolha do SGBD utilizado no projeto.

O capítulo 4 disserta sobre o SOA-MC e a parte teórica para o entendimento de *Web Services*, enquanto o capítulo 5 é centrado no *Web Service* desenvolvido para o projeto: quais tecnologias foram utilizadas e por que, quais seus métodos e como se deu o processo de criação do mesmo.

Os capítulos 6 e 7 abordam as aplicações clientes do Doador Sangue Bom, para *web* e para Android, respectivamente, explicando as tecnologias utilizadas, o motivo da escolha de cada uma delas e detalhando como cada cliente interage com o

Web Service e quais suas funcionalidades. Também é explicado como se deu o processo de publicação do aplicativo Android no Google Play.

O capítulo 8 apresenta as conclusões sobre este trabalho, enquanto o capítulo 9 apresenta possíveis expansões do projeto.

Capítulo 2 - O Doador Sangue Bom

2.1. Como Funciona

O Doador Sangue Bom consiste de um sistema para gestão e promoção de doações de sangue e pode ser acessado via navegador, pela *web* e por *smartphones* que executam o sistema operacional Android.

O sistema possui as seguintes funcionalidades básicas, disponíveis para o usuário normal (doador/receptor):

- Cadastro de usuário com nome, sobrenome, e-mail, gênero, data de nascimento e tipo sanguíneo.
 - A data de nascimento é importante porque usuários com menos de 18 anos e mais de 65 anos não são indicados a doar sangue.
 - O usuário é encorajado a cadastrar, também, o seu endereço, embora não seja obrigatório.
- Solicitação de doação de sangue.
- Visualização de doações disponíveis com seu respectivo motivo, o tipo sanguíneo do paciente e o local em que a doação está sendo aceita.
- Visualização de hospitais e das doações de cada um deles.
- Visualização de um mapa com a localização dos hospitais onde houve pedidos de doação e do hospital mais próximo da casa do usuário, caso o usuário possua um endereço cadastrado.
- Visualização de um hospital que aceita doações e esteja mais próximo de onde o usuário se encontra no momento.
 - Essa funcionalidade só é válida para o aplicativo Android, que utiliza os serviços de localização para saber onde o usuário se encontra no momento.
- Visualização de um histórico contendo as doações realizadas pelo usuário.
- Visualização de notícias pertinentes.

O usuário do Doador Sangue Bom pode ser de três tipos definidos pelos participantes do projeto. Esses tipos são chamados de atores do sistema e são: administrador, doador/receptor e coletor.

O administrador é o usuário com maior nível de privilégio dentro do sistema e suas principais funções são: cadastrar novos hospitais, lançar notícias pertinentes, gerar códigos de coletor e remover usuários do sistema por motivos como uso indevido. Ao gerar o código de coletor, o administrador deve associá-lo ao hospital em que esse coletor trabalha.

O doador/receptor é o usuário padrão do sistema e suas funções principais são: requisitar doações de sangue ou declarar a intenção de doar para uma pessoa. Após declarada, a intenção de doar é analisada e confirmada ou rejeitada por um usuário do tipo coletor.

O coletor representa um coletor de sangue, embora, apesar do nome, esse tipo de usuário possa ser qualquer pessoa escolhida pela gerência do setor de doações de sangue de um determinado hospital. A função principal de um coletor é validar intenções de doação de sangue, feitas pelos usuários. Ele tem a responsabilidade de indicar se uma doação foi realmente feita ou não. Um coletor é vinculado a um hospital e, ao se cadastrar, o coletor deve informar um código, fornecido por um administrador do sistema. Esse código tem o objetivo de validar a identidade do coletor e está associado ao hospital em que esse coletor trabalha.

A Figura 2.1 ilustra o processo de validação de doações realizado pelo coletor.

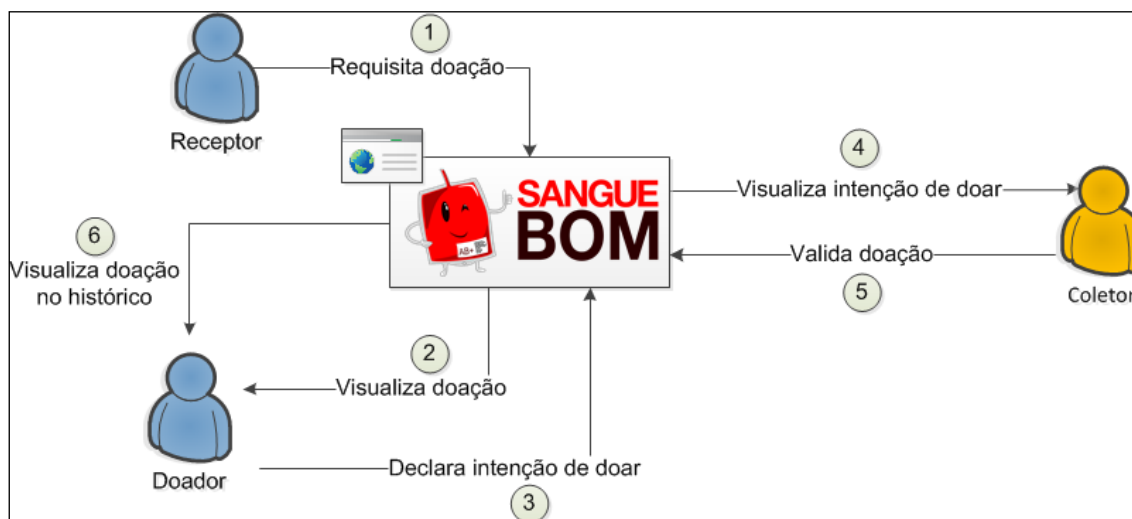


Figura 2.1. Processo de validação de doações de sangue

Os seis passos são explicados a seguir:

1. Um usuário denominado como “Receptor” requisita uma doação no sistema Doador Sangue Bom em nome de um indivíduo denominado “Paciente”.
2. Outro usuário, denominado “Doador”, visualiza a doação cadastrada pelo usuário “Receptor”.
3. O usuário “Doador” registra sua intenção de doar no sistema Doador Sangue Bom.
4. Um terceiro usuário denominado “Coletor” verifica que foi registrada uma intenção de doar por parte do usuário “Doador” no hospital em que o “Coletor” trabalha.
5. Após comparar os nomes dos usuários que registraram intenção de doar com os nomes das pessoas que efetivamente doaram para o “Paciente”, o “Coletor” confirma que o “Doador” realizou uma doação para o “Paciente” e a valida utilizando o sistema.
6. Após a validação, o histórico do “Doador” é atualizado com a doação realizada para o “Paciente”.

É importante lembrar que tanto coletores quanto administradores também podem utilizar todas as funcionalidades que um usuário doador/receptor é capaz. No entanto, um administrador não tem acesso às funcionalidades que um coletor tem, e vice-versa.

2.2. Inovação Social

O Brasil ainda não apresenta um número satisfatório de doadores de sangue regulares. Segundo parâmetros da OMS, o ideal seria que o número de doadores fosse o equivalente a 3% da população. Hoje, o número de doadores regulares no Brasil equivale a cerca de 1,8% da população [4].

Considerando apenas o estado do Rio de Janeiro, as pesquisas apontam para esse mesmo número. Foi constatado que em 2011 o índice de doadores em relação à população foi de 1,83%, como mostra o gráfico na figura 2.1 [5].

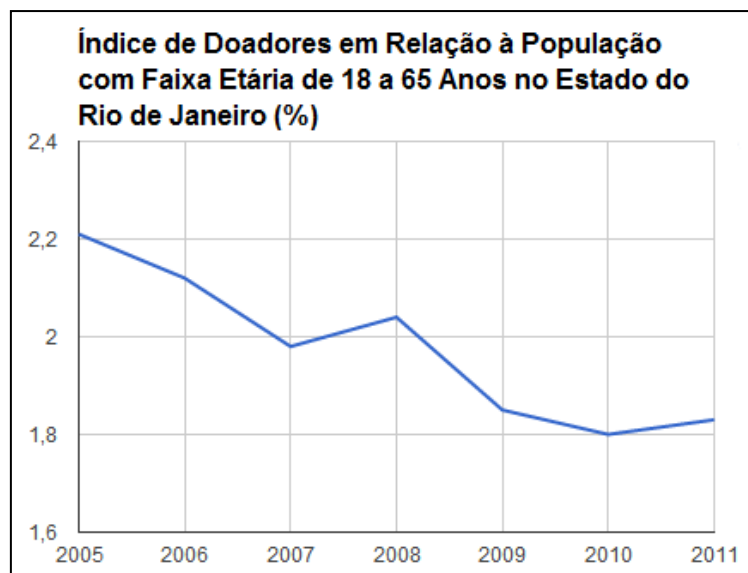


Figura 2.2. Índice de doadores em relação à população
Fonte: HEMOPROD e DATASUS

A situação do número de doadores é ainda mais agravante em períodos como o Carnaval e o inverno. Um dos motivos para essa queda é o fato de muitas pessoas viajarem nesses momentos do ano. No inverno, o frio e as chuvas são outros motivos que afetam a disponibilidade dos doadores. No Carnaval, o número de doações no Rio de Janeiro sofre queda de 25%. No inverno essa queda pode chegar a 40% [6] [7].

Para se ter uma ideia, o Hemorio, hemocentro público do Rio de Janeiro, recebe 300 voluntários por dia [4]. Em 2011, a hemorrede pública de todo o estado coletou 189.956 bolsas de sangue, como demonstra a figura 2.3, sendo que a meta é alcançar 300.000 doações [5].

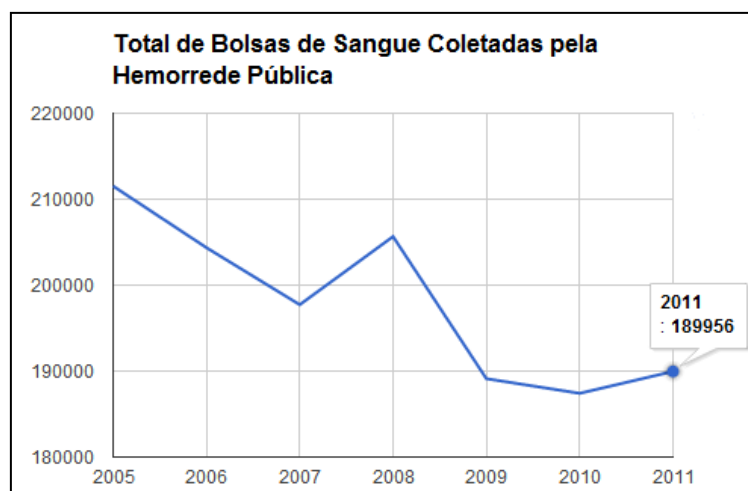


Figura 2.3. Total de bolsas de sangue coletadas pela hemorrede pública
Fonte: HEMOPROD

Apesar de os números ainda não serem satisfatórios, o Ministério da Saúde está investindo em recursos para ampliar o número de doações, utilizando-se de campanhas no rádio, em eventos esportivos e nas redes sociais [4]. O Doador Sangue Bom seria mais uma ferramenta que contribuiria para essa ampliação.

Segundo MULGAN [8], uma definição simples de inovação é: novas ideias que funcionam. Ainda segundo MULGAN [8], inovação social seria caracterizada como novas ideias que funcionam ao cumprir objetivos sociais. Diferentemente da inovação corporativa, que visa obter lucro, a inovação social visa obter o bem-estar das pessoas através de ideias.

Por se tratar de uma contribuição para a saúde pública através de um serviço inovador, tendo em vista seus recursos detalhados na seção 2.1, o Doador Sangue Bom pode ser caracterizado como uma inovação social.

Buscando aumentar a capacidade de uma sociedade para resolver um problema [8], uma das melhores alternativas é utilizar a colaboração. O sucesso do trabalho aqui proposto está atrelado à colaboração entre seus usuários. Sua facilidade de acesso e possibilidade de compartilhamento em redes sociais facilitará a construção de uma rede de colaboradores centralizados numa plataforma capaz de alterar o cenário de doações de sangue. A concentração inicial será no município do Rio de Janeiro, onde os autores residem. De acordo com a popularidade do sistema, serão estudadas formas de expansão.

Capítulo 3 - O banco de dados do Doador Sangue Bom

Os dados do Doador Sangue Bom consistem de informações acerca das doações, usuários e hospitais, além de outras características importantes para o sistema. Esses dados devem ficar armazenados e organizados em um banco de dados, para que possam ser acessados das aplicações clientes. Para tal, fez-se necessária a escolha de um Sistema de Gerenciamento de Banco de Dados (SGBD) para o sistema em questão.

3.1. A escolha do SGBD

Existem diversos tipos de SGBD disponíveis e, durante a graduação, tivemos contato com alguns deles, sendo a maioria deles relacional e com linguagem de consulta estruturada (SQL). Ao escolher um SGBD adequado para o problema, é importante fazer algumas perguntas essenciais que auxiliam na escolha da melhor opção:

- Qual o volume de dados estimado?
- Quais os tipos de dado armazenados?
- Qual a frequência de acesso ao banco de dados?

Dadas as perguntas, foi o escolhido o SGBD mais adequado para o sistema de acordo com as respostas. Como há diversas opções de escolha disponíveis, limitamos a escolha do SGBD baseando-se nos seguintes critérios: banco de dados relacional com SQL, experiência prévia e *software* proprietário. A utilização de um banco de dados relacional que utilize SQL mostrou-se bastante óbvia neste tipo de projeto, em que a relação entre as diversas partes é essencial, facilitando, além disso, a forma como as consultas são realizadas e os dados são recuperados do banco de dados. Com a lista limitada, finalmente, pudemos responder as perguntas acima e decidir qual a melhor escolha. A lista de SGBD limitada foi:

- MySQL
- SQLServer
- PostgreSQL

O SQLServer é um banco de dados da Microsoft e, embora possua licença, sua versão gratuita permite o armazenamento de até 10 GB de dados. O PostgreSQL pertence ao PostgreSQL Global Development Group, é gratuito e possui código aberto, assim como o MySQL, que pertence à Oracle.

Dadas as três opções e as perguntas acima, as respostas permitiram escolher qual o SGBD mais adequado para o Doador Sangue Bom.

O volume de dados é uma característica difícil de ser estimada, já que depende do sucesso e da escalabilidade do sistema. Como a escalabilidade é sempre desejável, estimamos que o volume de dados pode ser grande e, com isso, excluimos o SQLServer da lista, já que sua versão gratuita possui limitação de espaço.

Quanto aos tipos de dados armazenados pelo Doador Sangue Bom, eles são, principalmente, *strings* e inteiros, demonstrando uma baixa complexidade.

Para a estimativa da frequência de acesso ao banco de dados, seguimos a mesma lógica utilizada na estimativa do volume de dados: embora, inicialmente, o sistema dependa muito de altruísmo e sua frequência de acesso não deva ser alta, é sempre desejável um sistema com muitos usuários, além do crescimento ser um dos objetivos principais do Doador Sangue Bom. Por isso, é de suma importância a escolha de um SGBD capaz de lidar bem com uma alta frequência de consultas ao banco de dados.

Tanto o MySQL e o PostgreSQL foram escolhas recomendadas pelo orientador. O MySQL é um SGBD largamente utilizado em aplicações que utilizam a *web*, além de possuir uma alta flexibilidade e um desempenho excelente. O PostgreSQL também é famoso pelo seu desempenho e pela sua flexibilidade, além da comunidade que o utiliza estar crescendo rapidamente [9].

Devido à fama do MySQL em aplicações web, sua flexibilidade e desempenho e a baixa complexidade do banco de dados do Doador Sangue Bom, optamos por utilizar o MySQL como SGBD do Doador Sangue Bom.

3.2. Modelagem

A modelagem do banco de dados do Doador Sangue Bom foi feita pelos dois membros do projeto, já que o entendimento de suas entidades e relacionamentos é essencial para desenvolvimento de todas as aplicações clientes, assim como do *Web Service*.

Para a modelagem do banco de dados e a inserção de dados pré-definidos utilizamos o MySQL *Workbench*, ferramenta de gerenciamento com diversas

funcionalidades relativas ao SGBD, como backup de banco de dados, execução de *scripts*, visualização de dados e geração de modelos.

O banco de dados foi modelado em inglês e possui treze entidades (representadas por tabelas no banco de dados): *actor*, *address*, *blood_type*, *collector*, *donation_disposal*, *donation_request*, *doua*, *history*, *hospital*, *news*, *resource*, *state* e *user*. Essas tabelas correspondem, respectivamente a: ator, endereço, tipo sanguíneo, coletor, intenção de doação, pedido de doação, controle de acesso, histórico, hospital, notícias, recursos, estado e usuário. A Figura 3.1 mostra o diagrama entidade-relacionamento do Banco de Dados do Doador Sangue Bom, gerado com o MySQL *Workbench*.

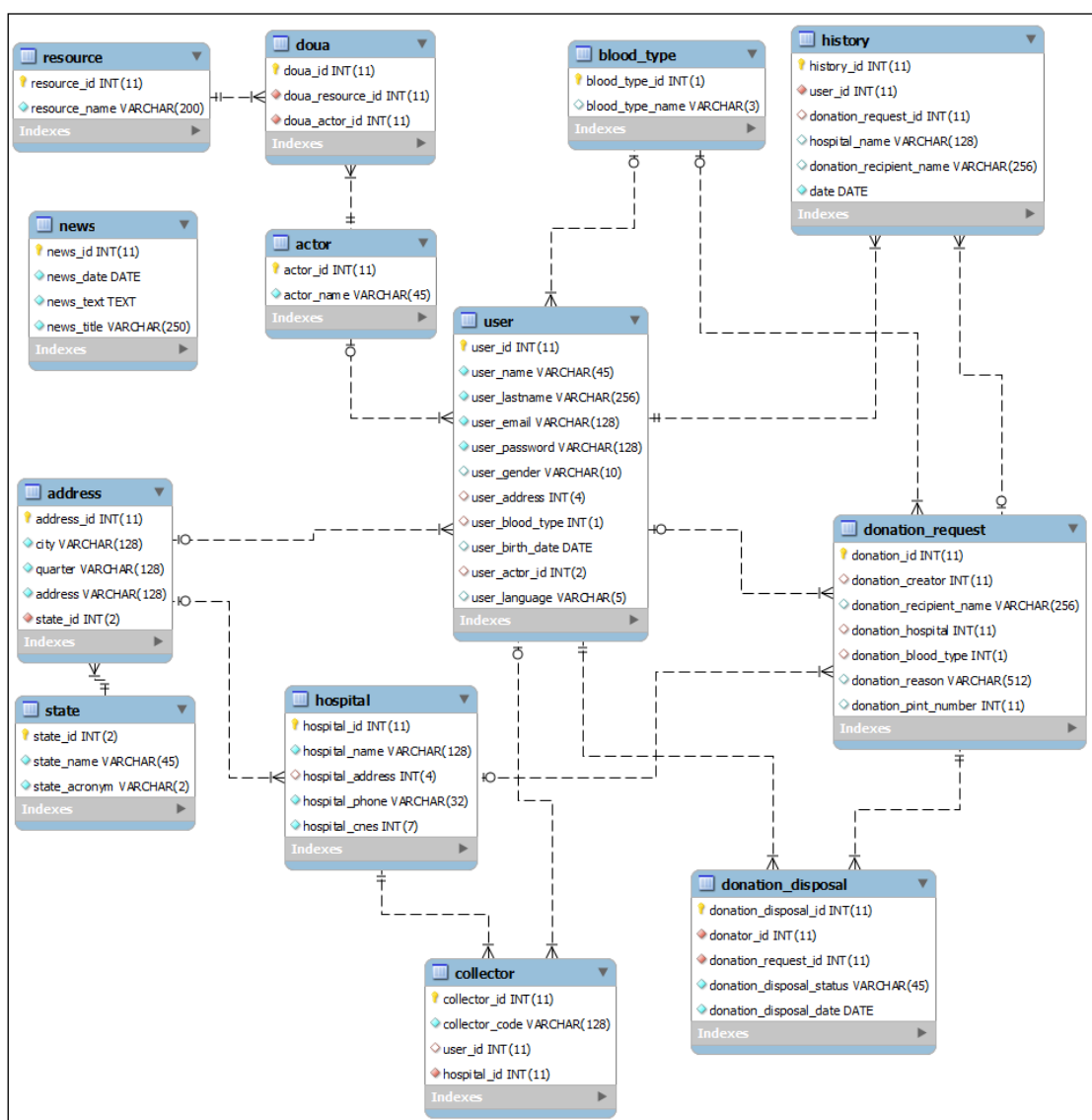


Figura 3.1. Modelo do banco de dados

Abaixo segue uma breve explicação de cada entidade do sistema:

Actor: define os atores do sistema, explicados no Capítulo 2. Caso haja necessidade de algum novo tipo de ator no futuro, basta adicioná-lo ao banco de dados.

Address: armazena os endereços utilizados no sistema, tanto o endereço de usuários quanto o de hospitais. Possui cidade, bairro e o endereço em si, além de fazer referência à lista de estados brasileiros, que fica armazenada na tabela *State*.

Blood_type: armazena os tipos sanguíneos.

Collector: Armazena os códigos de coletor que serão utilizados por usuários desse tipo no momento do cadastro. O administrador gera um código e define para qual hospital ele é vinculado. Quando um coletor de determinado hospital cadastra-se no sistema, ele é associado ao código para que seja possível identificá-lo.

Donation_Disposal: Armazena as intenções de doação. Faz referência a um usuário, o que declarou a intenção de doar e a um pedido de doação. Também indica se aquela intenção foi validada ou não por um coletor.

Donation_Request: Armazena as doações que foram pedidas pelo sistema. Faz referência a um usuário que a criou, a um tipo sanguíneo e a um hospital. Também possui campos para declarar um motivo para a doação, para o nome do paciente em questão, além da quantidade de bolsas de sangue necessárias, caso aplicável.

Doua: Tabela utilizada para mapear casos de uso com grupos de usuário. Seu funcionamento será melhor explicado na seção 3.3.

History: Faz referência a um usuário e a um pedido de doação. Também é possível armazenar doações passadas que não foram feitas por intermédio do sistema.

Hospital: Armazena os hospitais cadastrados no sistema, com seus respectivos telefones e CNES [10], além de fazer referência a um endereço.

News: Armazena as notícias cadastradas no sistema.

Resource: Casos de uso do sistema. Cada função não pública do *Web Service* possui uma entrada nessa tabela. Seu uso será melhor explicado na seção seguinte.

State: Armazena todos os estados do Brasil.

User: Armazena as informações pertinentes ao usuário: nome, sobrenome, tipo sanguíneo, e-mail, senha, gênero e data de nascimento. Faz referência à um endereço e a um tipo de usuário, definido na tabela de atores do sistema.

3.3. DOUA - *Database Oriented Usecase Authorization*

No desenvolvimento de sistemas existem dois conceitos de extrema importância: autenticação e autorização de usuários. O processo de autenticação consiste em verificar se um usuário é realmente quem ele diz ser e, na maioria dos sistemas atuais, isso ocorre através de um *login* e uma senha. A autorização, por outro lado, requer que já tenha ocorrido uma autenticação e consiste em definir quais recursos, dada uma lista, um usuário pode acessar.

DOUA consiste de uma camada de um sistema que lida com toda a parte de autorização. Segundo a filosofia do DOUA, um sistema possui atores e casos de uso (ou recursos). O ator de um sistema é um grupo de usuários que compartilham as mesmas características, enquanto um caso de uso consiste de uma operação ou método que um determinado usuário pode executar no sistema. Atores e casos de uso estão relacionados entre si em uma relação de muitos para muitos.

Para o DOUA, é importante que a informação dos atores e dos casos de uso sejam carregadas de uma fonte de dados externa, podendo ser um arquivo XML, JSON ou um banco de dados. Com essa característica, melhora-se a manutenibilidade do sistema, já que é preferível alterar dados a códigos, devido ao baixo custo.

O Doador Sangue Bom utiliza o DOUA para implementar a autorização dos grupos de usuário. Os três grupos de usuário do sistema são: administrador, doador/receptor e coletor, e suas características já foram explicadas no Capítulo 2. Os casos de uso do Doador Sangue Bom são as funções não públicas do sistema. Toda função que requer uma autenticação prévia é considerada uma função não pública. Por exemplo, funções como o cadastro de usuário ou o *login* podem ser executadas sem a necessidade prévia de uma sessão de usuário. Optamos por não incluir essas funções nos casos de uso do sistema e, portanto, elas não passam pelo processo de autorização.

O DOUA desempenha um papel importante para sistemas que utilizam o SOA-MC, como é o caso do Doador Sangue Bom, já que, os múltiplos clientes podem usufruir da camada de autorização com poucas mudanças em códigos-fonte.

Na Figura 3.2 está o modelo do DOUA utilizado no Doador Sangue Bom. A tabela *doua* relaciona-se com os casos de uso e com os atores e é acessada sempre que o sistema deseja verificar se algum usuário possui autorização.

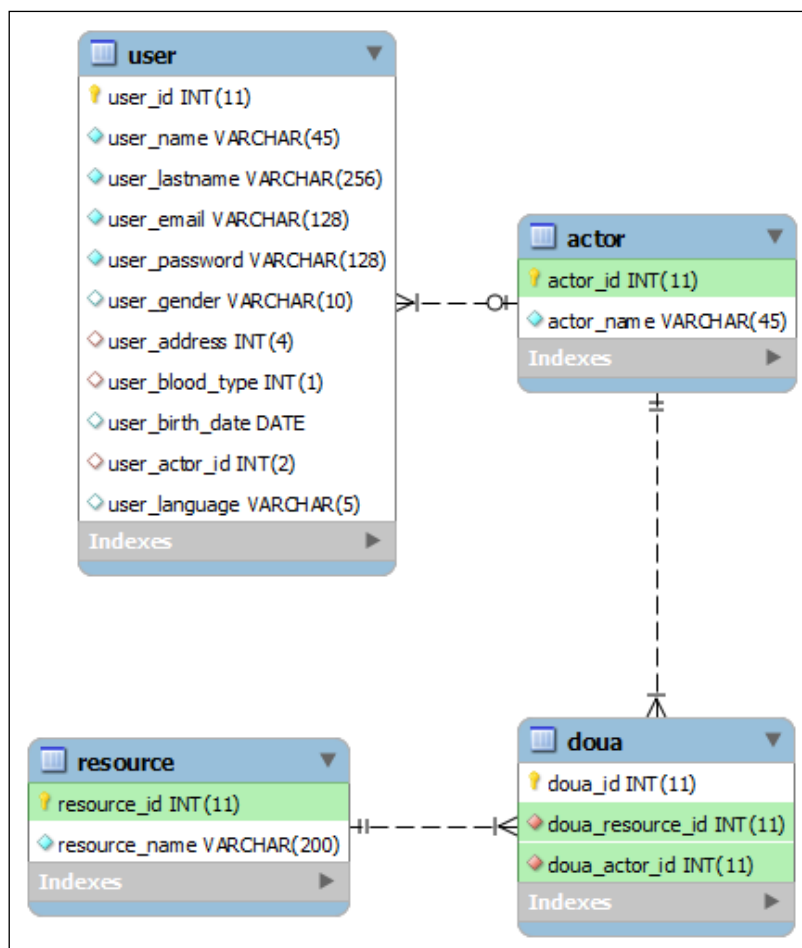


Figura 3.2. Modelo do DOUA

Capítulo 4 - *Web Services* e SOA-MC - Teoria

Ao desenvolver sistemas interdependentes, é sempre essencial a escolha de uma arquitetura que possa integrá-los com a menor quantidade de retrabalho, de gastos e que torne os sistemas facilmente manuteníveis. Atualmente, a demanda por aplicações distribuídas cresceu visivelmente e, com isso, a necessidade de uma maior integração entre diversas aplicações.

A escolha da arquitetura ideal para um determinado sistema depende dos requisitos e da complexidade dele. É desejável a escolha de uma arquitetura que diminua a complexidade de desenvolvimento, agilizando-o e facilitando a integração entre as partes do sistema.

A tendência no desenvolvimento de sistemas tem sido a criação de serviços para descrever suas funcionalidades. A capacidade de ocultar seu funcionamento de ambientes externos, de apresentar uma interface simples de acesso, de promover a reutilização e de concentrar o comportamento de um sistema são apenas algumas vantagens na utilização de serviços para elaboração de projetos de *software*.

4.1. Arquitetura Orientada a Serviços - Múltiplos Clientes

Existem diversos tipos de arquiteturas que propõem soluções para os problemas encontrados no desenvolvimento de sistemas. Neste trabalho, optamos pela utilização de uma arquitetura orientada a serviços (SOA), mais especificamente uma arquitetura orientada a serviços com múltiplos clientes (SOA-MC).

O projeto do Doador Sangue Bom consiste de duas aplicações distintas: um cliente *web* e um cliente *smartphone*. A forma como esses clientes foram desenvolvidos são diferentes e, embora neste trabalho ambos utilizem a linguagem de programação Java, são sistemas que serão executados em plataformas diferentes. Portanto, é necessária uma arquitetura que preserve a independência entre os sistemas, mesmo que ambos tenham acesso aos mesmos recursos e dados.

A definição de uma arquitetura orientada a serviços, dada pela empresa Everware-CBDI é de que SOA consiste nas políticas, práticas e estruturas que permitem que as funcionalidades de uma aplicação sejam providas e consumidas como um grupo de serviços publicados com uma granularidade relevante ao

consumidor do serviço. Esses serviços podem ser invocados, publicados e descobertos e são abstraídos da implementação utilizando certo padrão de interfaces [11] [12].

Logo, uma arquitetura orientada a serviços deve prover uma interface independente da plataforma, permitindo o acesso às suas funcionalidades a partir de ambientes e plataformas completamente diversos e diferenciados. As mensagens enviadas através dessa interface devem ser descritivas e não instrutivas [13], ou seja, elas devem apenas descrever o que deve ser feito com quais parâmetros, qual serviço está sendo requisitado e qual o resultado esperado. Uma interface não deve conter inteligência sobre como o processo vai ser feito ou implementado. Dessa forma, as aplicações clientes sabem exatamente “o que pedir” através das mensagens enviadas e o serviço sabe exatamente “o que prover” e “como fazer”.

Outra característica desse tipo de arquitetura é a promoção da reutilização. Refazer funções e métodos que já foram desenvolvidos pode acarretar em perda de tempo e crescimento desnecessário do código, causando prejuízos para determinada organização ou desenvolvedor. Com o uso do SOA, é possível reutilizar funcionalidades em sistemas completamente distintos e com objetivos totalmente diferentes. As regras de negócio são implementadas apenas uma vez e podem ser utilizadas em qualquer cliente.

Além dos provedores do serviço e dos consumidores do serviço (os clientes), esse tipo de arquitetura deve possuir um diretório de serviços, que especifica quais os serviços disponíveis para consumo [14]. Um consumidor deve checar esse diretório e descobrir quais serviços ele pode utilizar, enquanto o provedor deve publicar seus serviços nesse diretório para que possam ser descobertos pelos consumidores. A Figura 4.1 ilustra o uso do diretório de serviços.

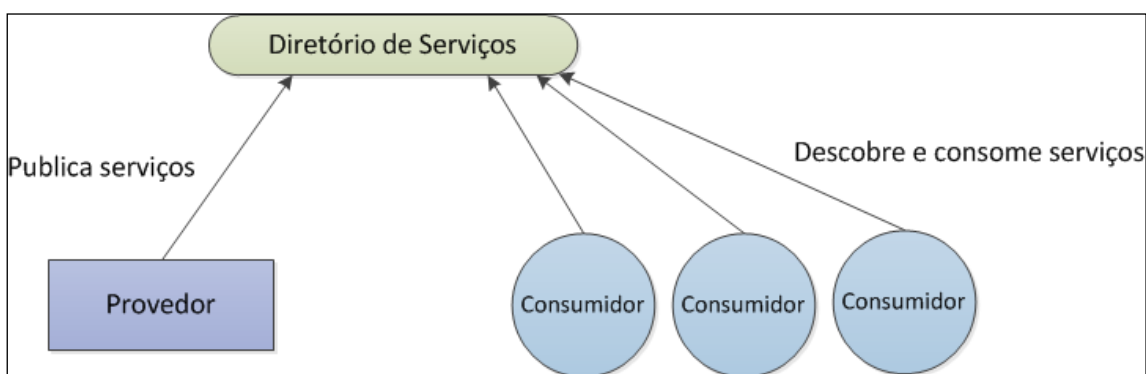


Figura 4.1. Diretório de serviços

A escalabilidade é alcançada facilmente com esse tipo de arquitetura, já que mudanças nos serviços não implicam em alterações na forma como esse serviço é

acessado. Isso se dá graças ao uso do envio de mensagens para a comunicação. No entanto, caso a interface, os parâmetros ou o retorno esperado sejam alterados, a comunicação entre os sistemas pode ser prejudicada.

A Figura 4.2 ilustra o SOA-MC. A utilização de *Web Services* não é obrigatória ao utilizar essa arquitetura, embora seja a forma mais utilizada. Pela figura, podemos notar que as regras de negócio são todas implementadas em uma das camadas, a que vai prover os serviços. Esse provedor enviará, então, mensagens para os consumidores (os clientes) com o resultado de alguma requisição feita anteriormente. As mensagens trocadas costumam possuir três formatos: XML, JSON e CSV, sendo o XML a mais utilizada.

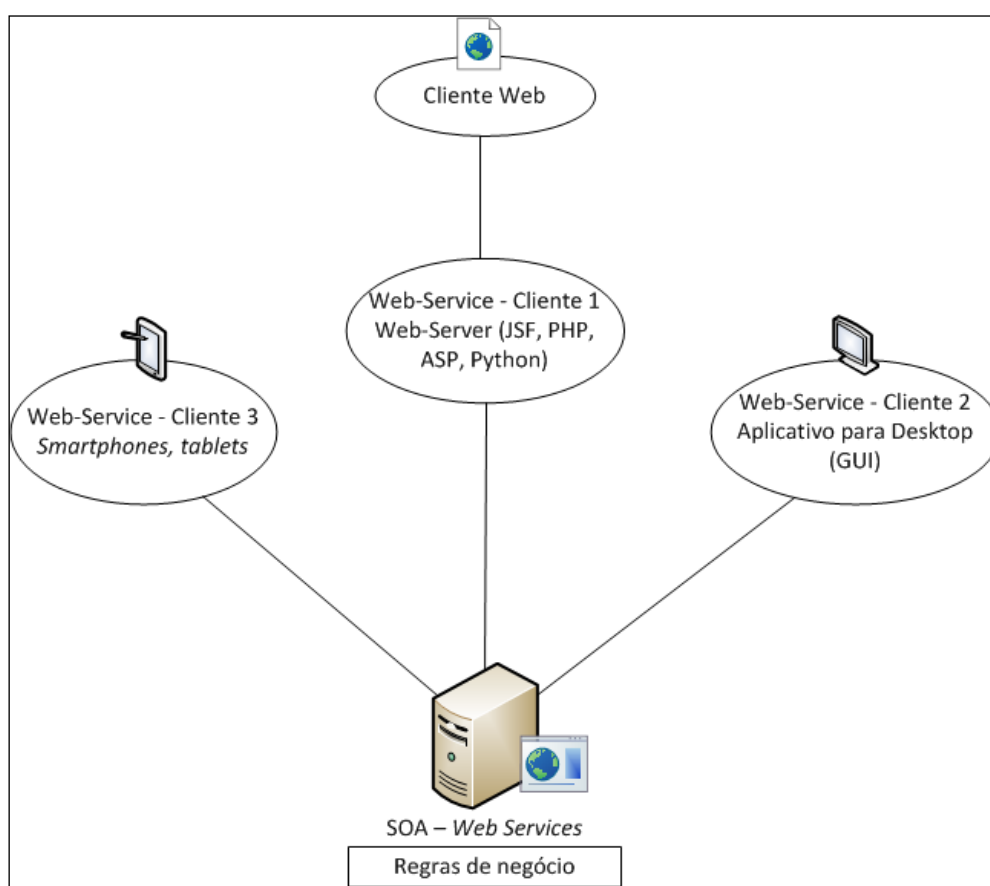


Figura 4.2. SOA-MC

4.2. Web Services

A definição do termo *Web Services* é muito debatida, mas é popularmente aceito que um *Web Service* consiste de uma camada de *software* baseada em uma arquitetura orientada a serviços em que as interfaces são baseadas em protocolos da

Internet (HTTP, SMTP, FTP). Além disso, as mensagens enviadas podem ser em XML ou JSON. Essa última apresenta crescente popularidade, embora a primeira seja a mais utilizada e difundida [15].

O XML é um formato de documento especificado pela W3C [16] que une um conjunto de regras para registrar dados de uma forma simples. Ele consiste de diversas entidades, que são unidades que possuem dados, e foi desenvolvido para ser simples, facilmente inteligível pelo ser humano, suportado por muitos tipos de aplicações diferentes e apresentar neutralidade entre sistemas. Assim, uma aplicação que utiliza XML para comunicação não está restrita a uma plataforma específica e independe do sistema operacional em que a aplicação está sendo executada, a linguagem em que ela foi desenvolvida ou o banco de dados onde estão armazenados seus dados. Além disso, a organização dos dados no XML depende apenas do desenvolvedor, dando maior liberdade na criação de aplicações que dependem dele. Outra vantagem do XML é que ele dá suporte ao Unicode, permitindo que caracteres de qualquer alfabeto sejam reconhecidos [17].

Devido a essas características, o XML tornou-se amplamente utilizado na integração de sistemas, já que um documento XML é escalável e facilmente lido por aplicações sendo executadas em plataformas diferentes. Ele também se tornou uma ferramenta essencial em sistemas que utilizam SOA, sobretudo na troca de mensagens entre *Web Services* e seus clientes.

Existem duas categorias de *Web Services* que se destacam: serviços que utilizam o protocolo SOAP e serviços *RESTful*. Ao desenvolver um *Web Service*, é preciso escolher qual das duas categorias será utilizada.

O SOAP é um protocolo de troca de mensagens entre aplicações distribuídas. As mensagens são codificadas em XML e utilizam um protocolo da *Internet* para serem transportadas.

As mensagens SOAP são encapsuladas em um envelope que, na prática, é uma *tag* XML. Dentro do envelope SOAP há um cabeçalho opcional e o corpo da mensagem. O cabeçalho contém informações acerca das mensagens e atributos que possam ser mandatórios para que alguma aplicação entenda a mensagem, como, por exemplo, alguma informação sensível indicando que um cliente está autorizado a receber os dados ou que um pagamento foi feito [18]. O corpo da mensagem contém a mensagem em si. Pode haver também, dentro dele, um elemento de falha para indicar erros encontrados na troca de mensagens e seus respectivos motivos. A Figura 4.3 mostra o XML, tanto da requisição quanto da resposta, de uma mensagem enviada pelo protocolo SOAP, contendo uma função de exemplo chamada *PegaSiglaPaís*, que retorna a sigla de um país.

```

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:s="http://www.exemplo.com">
  <env:Body>
    <s:pegaSiglaPaís>
      <s:País>Brasil</s:País>
    </s:pegaSiglaPaís>
  </env:Body>
</env:Envelope>

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:s="http://www.exemplo.com">
  <env:Body>
    <s:PegaSiglaPaísResponse>
      <s:Sigla>BR</s:Sigla>
    </s:PegaSiglaPaísResponse>
  </env:Body>
</env:Envelope>

```

Figura 4.3. Mensagem SOAP

Tratando-se de *Web Services* que utilizam SOAP, sempre há um WSDL agregado. O WSDL é uma linguagem, definida em XML, que descreve os serviços disponíveis e também aponta a localização dos mesmos [19]. Com o WSDL, uma aplicação cliente sabe quais métodos ela pode utilizar, quais parâmetros de cada método devem ser passados e qual o retorno esperado pelo serviço.

A Figura 4.4 mostra o esqueleto de um WSDL. A *tag definitions* é a raiz do documento e define o nome do *Web Service*, assim como seus *namespaces*. Dentro desse elemento estão todos os outros elementos do WSDL. Em *types* ficam definidos os tipos de dados que serão utilizados nas mensagens. Já a *tag messages* é uma representação abstrata das mensagens que serão trocadas, tanto das requisições como das respostas. A *tag portType* contém um conjunto de operações, que é uma definição abstrata dos métodos do *Web Service* e em *binding* define-se o protocolo e formato dos dados para as operações. Por exemplo, é a *tag binding* que define a versão do SOAP que será utilizada na troca de mensagens. Por fim, a *tag service* é uma lista dos serviços definidos no WSDL, de acordo com os protocolos utilizados.

```

<definitions>
  <types>
    Definição dos tipos.
  </types>

  <message>
    Definição de uma mensagem.
  </message>

  <portType>
    <operation>
      Definição de uma operação.
    </operation>
  </portType>

  <binding>
    Definição de uma ligação.
  </binding>

  <service>
    Definição de um serviço.
  </service>
</definitions>

```

Figura 4.4. Esqueleto de um WSDL

A Figura 4.5 mostra um exemplo: um trecho do WSDL do Doador Sangue Bom. Nela podemos ver a descrição da função *saveAddress*, para salvar um endereço no banco de dados. Há quatro parâmetros exigidos: a cidade, o estado, o bairro e o endereço em si. Logo abaixo, está especificado o formato da resposta retornada pelo *Web Service*, após a execução dessa função.

```

<xs:element name="saveAddress">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="address" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="city" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="state" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="quarter" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="saveAddressResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figura 4.5. WSDL

Também é importante citar o UDDI, que é um padrão, definido por um documento XML, que lida com a publicação e a descoberta de *Web Services* e está associado ao WSDL. O UDDI funciona como um mediador do serviço e pode possuir

definições de taxonomia para categorizar serviços e referências a especificações que um determinado *Web Service* suporta [20].

Dentre as principais vantagens do uso do protocolo SOAP pode-se citar: é independente de plataforma e linguagem; é em XML; possui o WSDL para descrever serviços e o UDDI para facilitar a descoberta de serviços; possui tratamento de erro embutido na mensagem; há facilidade de desenvolvimento de aplicações clientes, já que existem muitas ferramentas que são capazes de gerar uma interface de comunicação a partir de um WSDL.

A Figura 4.6 ilustra a troca de mensagens SOAP.

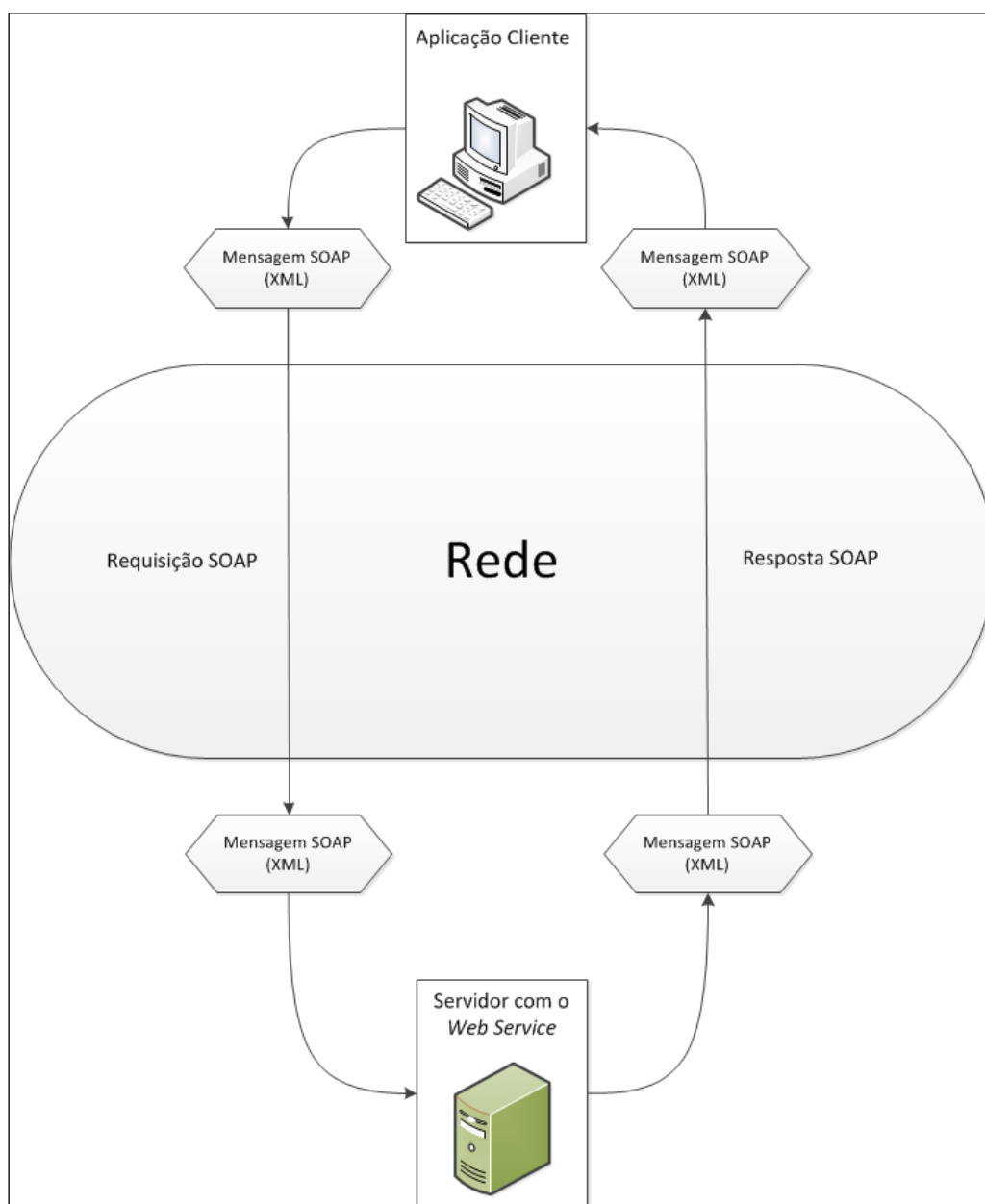


Figura 4.6. Troca de mensagens SOAP

No entanto, o SOAP também possui algumas desvantagens como a maior dificuldade de entendimento ou a necessidade de mais recursos computacionais para

ser processado. Ambas desvantagens são válidas apenas se compararmos o SOAP com serviços *RESTful*. Além disso, *Web Services* que utilizam SOAP são mais difíceis de ser implementados, criando a necessidade do uso de ferramentas para a geração do WSDL e do processamento do XML.

Outra categoria de *Web Services* que vêm crescendo largamente são os que utilizam o modelo REST, chamados de serviços *RESTful*. O surgimento dessa categoria foi uma reação aos padrões do SOAP, que faziam da troca de mensagens um processo relativamente dispendioso.

A simplicidade do modelo REST reside no fato de que um *Web Service* é tratado como um recurso, isto é, pode ser identificado por uma URI. As mensagens trocadas com o modelo REST podem ser enviadas apenas via HTTP e utilizam suas primitivas para a manipulação dos recursos. Essas primitivas são: GET, PUT, POST e DELETE. Um dos princípios do modelo REST é estabelecer um mapeamento entre as primitivas do HTTP e as operações básicas de CRUD (Criar, Ler, Atualizar e Remover) [21] [22]. Com esse mapeamento:

- POST é responsável por CRIAR um recurso.
- GET é responsável por LER um recurso.
- PUT é responsável por ATUALIZAR um recurso.
- DELETE é responsável por REMOVER um recurso.

A Figura 4.7 exemplifica, de forma genérica, algumas requisições a um *Web Service* implementado com REST.

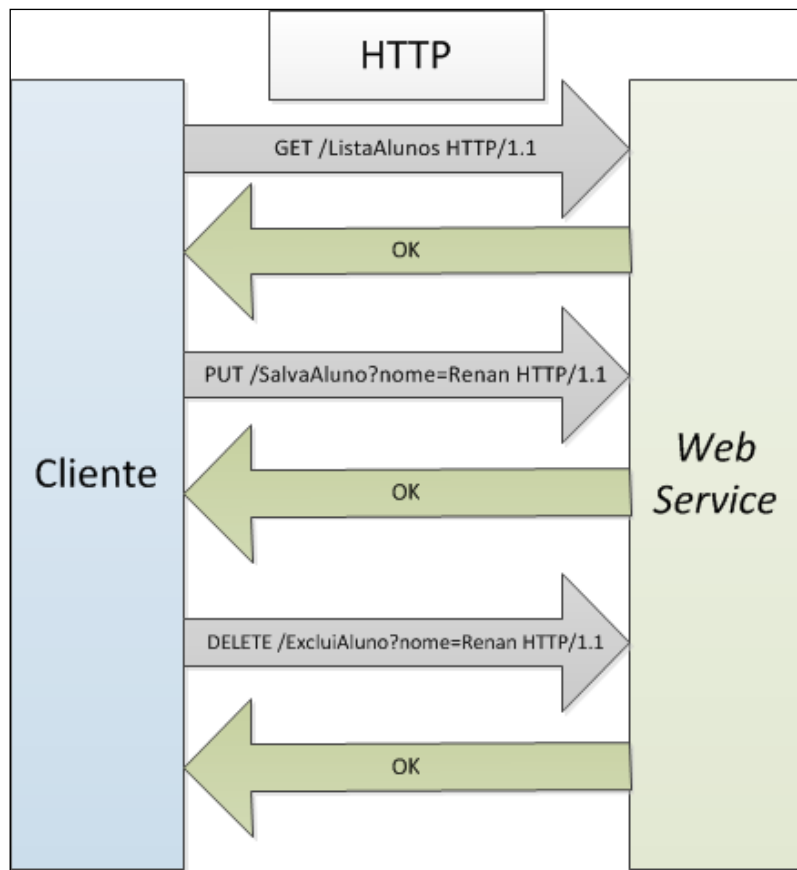


Figura 4.7. Requisições em um serviço *RESTful*

Em um serviço que usa REST, é importante que não haja nenhuma informação sobre estado no servidor, indicando que não há nenhum tipo de dado relativo à sessão do cliente. Todas as requisições do cliente devem conter todas as informações necessárias para que o servidor as interprete [23]. Essa característica do modelo aumenta a escalabilidade dos *Web Services* e simplifica a implementação de serviços, já que os recursos podem ser liberados mais facilmente, sem a necessidade de nenhuma checagem de estado.

As características supracitadas fazem do modelo REST muito semelhante à própria *Web*, o que é uma vantagem. Apesar disso, o modelo também possui desvantagens:

- Não é aconselhável enviar dados que devam ser protegidos, como parâmetros de uma URI.
- O SOAP lida bem com qualquer tipo de arquivo: texto ou binário. Já o REST pode ter problemas ao lidar com alguns tipos de arquivos, como binários, já que os dados são enviados através de uma URI.
- É vinculado ao protocolo HTTP, enquanto o SOAP pode utilizar outros protocolos para comunicação.

Apesar do modelo REST estar ganhando espaço ultimamente, o SOAP sempre foi a categoria de *Web Services* mais utilizada.

Dadas as vantagens e desvantagens dos dois modelos, reuniões com o orientador definiram que seria mais adequado utilizar o protocolo SOAP no Doador Sangue Bom, pois, além das vantagens expostas, a exposição de serviços, ao invés de dados, mostrou-se mais interessante, além de permitir o aprendizado no uso de ferramentas como o Axis2.

Capítulo 5 - O *Web Service* do Doador Sangue Bom

Neste capítulo será apresentado o *Web Service* do Doador Sangue Bom, suas funcionalidades, suas principais características e quais ferramentas foram utilizadas no seu desenvolvimento.

O *Web Service* foi desenvolvido em conjunto pelos dois membros do projeto, já que suas funcionalidades são importantes independente do cliente. Novos métodos foram elaborados sempre que necessário.

5.1. Tecnologias utilizadas

A escolha minuciosa das tecnologias utilizadas em um projeto é de suma importância, já que todo o desenvolvimento dependerá das ferramentas escolhidas previamente e uma mudança ao longo do desenvolvimento não é aconselhável, podendo gerar atrasos e problemas desnecessários.

Para o desenvolvimento de *Web Services* utilizando o protocolo SOAP, existem diversas linguagens e ferramentas úteis. A escolha da linguagem de programação foi praticamente imediata. O uso do protocolo SOAP em *Web Services* vêm quase sempre acompanhado da linguagem Java [24] e esta foi a linguagem escolhida para o desenvolvimento do *Web Service* do Doador Sangue Bom.

É importante também escolher um IDE para desenvolvimento. Dentre os mais conhecidos estão o Netbeans e o Eclipse. Ambos os IDEs possuem muitas semelhanças e diferenças e a escolha de um IDE melhor para o desenvolvimento é quase subjetiva. Optamos pelo Netbeans para a criação do *Web Service* devido ao seu *plugin* do Axis2, que é considerado mais estável que a versão do Eclipse.

O Axis2 [25] é uma ferramenta para desenvolvimento de *Web Services* e possui versões para as linguagens Java e C, assim como suporte para serviços que utilizam SOAP ou REST. A versão utilizada no Doador Sangue Bom é a versão para Java e que trabalha com o protocolo SOAP. Seu *plugin* para o Netbeans facilita a criação de *Web Services* e permite que todo o processo seja feito dentro da IDE, sem a necessidade da utilização de linhas de comando. O Axis2 também lida com toda a parte de criação do envelope das mensagens SOAP, elaboração do WSDL e o envio das mensagens pela rede.

Há várias características do Axis2 que tornam seu uso atraente. O processamento do XML das mensagens SOAP é feito de forma eficiente pela

ferramenta, utilizando uma representação de mensagens própria, conhecida como Axiom. Essa característica é crucial em um *Web Service*, já que o processamento da mensagem é um fator que incide sobre todo o desempenho de um sistema. O Axis2 também lida com a parte de implantação de um *Web Service*, tornando o processo mais simples e configurável pelo desenvolvedor, permitindo também que implantações sejam feitas assim que um novo serviço é criado, sem a necessidade de reiniciar o servidor. Por fim, o transporte das mensagens também é configurável, permitindo que elas sejam transmitidas via HTTP, SMTP, dentre outros protocolos de transporte, inclusive elaborados pelo próprio desenvolvedor [26].

A ferramenta também tem a capacidade de geração de código. Após gerado o código de um serviço, toda a parte de comunicação e interpretação de mensagens fica parcialmente visível para o desenvolvedor, que pode se ater simplesmente à implementação dos serviços.

O servidor escolhido para hospedar o *Web Service* foi o Apache Tomcat 7 [27], porque ele possui integração com o Netbeans, além de ser largamente utilizado para o desenvolvimento de aplicações Java. Além disso, dada a experiência prévia dos membros do projeto em disciplinas da graduação, o desempenho do Tomcat mostrou-se melhor em relação a outros servidores disponíveis.

A Tabela 5.1 contém a descrição de todas as ferramentas utilizadas no desenvolvimento do *Web Service*:

Ferramenta	Explicação
Netbeans IDE 7.0.1	O IDE para desenvolvimento do <i>Web Service</i> .
Java SE 6 Development Kit (JDK)	Kit de desenvolvimento para escrever aplicações em Java.
Apache Tomcat 7.0.14	Servidor para hospedagem do serviço.
Apache Axis2/Java 1.6.1	Ferramenta para criação de <i>Web Services</i>

Tabela 5.1. Ferramentas utilizadas para desenvolvimento do *Web Service*.

5.2. Criação do *Web Service*

Para a criação do *Web Service* do Doador Sangue Bom foi necessário decidir quais funções seriam necessárias para a elaboração do sistema, levando em conta tanto o cliente *Web* quanto o cliente *Android*.

Os membros do projeto se reuniram para decidir quais métodos deveriam ser implementados e qual o nível de autorização de cada um deles. No entanto, esses

métodos sofreram, ao longo do tempo, mudanças como o escopo de alguma funcionalidade ou o surgimento de uma nova necessidade.

O *Web Service* do Doador Sangue Bom concentra toda a conexão com o banco de dados e utiliza-se do JDBC do MySQL para a conexão. Foram elaboradas classes de *Data Access Object* [28] para acessar o banco de dados. Um *Data Access Object* (DAO) é um padrão de projeto (*design pattern*) de acesso aos dados em uma determinada fonte de dados, que pode ser tanto um banco de dados como um arquivo de texto ou um XML. O objetivo desse objeto é isolar toda a parte de conexão à fonte de dados, de forma que todo o acesso seja abstraído do restante do projeto, que poderá acessar esses objetos independente de como o acesso à fonte de dados foi implementado.

Uma classe chamada *DAOConnect* foi criada para acessar o banco de dados MySQL utilizando um JDBC. Essa classe possui informações como o endereço do banco de dados, a porta utilizada, o nome do banco e informações de autorização como nome de usuário e senha. Todos os outros DAOs utilizam essa classe para se conectar e existe um DAO para cada uma das tabelas do banco de dados do sistema.

Todas as senhas de usuários são armazenadas encriptadas no banco de dados e é o *Web Service* o responsável pela encriptação. Um algoritmo de *hash* de 128 *bits* é utilizado sempre que um usuário acessa o sistema e a senha encriptada é comparada com o valor salvo no banco de dados. Isso significa que, quando a senha é gerada, um algoritmo a transforma em uma sequência de números hexadecimais formada por 128 *bits* (cada número hexadecimal corresponde a 4 *bits*). O algoritmo utilizado foi o MD5 (*Message Digest Algorithm*). Para exemplificar o seu funcionamento, a Figura 5.1 demonstra como a senha “doadorsanguebom” seria salva no banco.

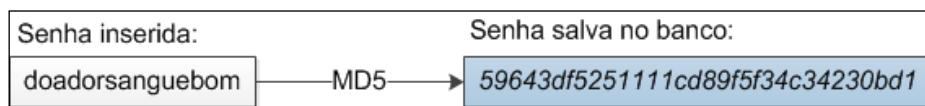


Figura 5.1. Senha em hash

Em cada DAO, as consultas e atualizações foram feitas utilizando um *PreparedStatement*, uma classe Java que permite elaborar uma expressão SQL pronta, pré-compilada, cujos parâmetros são passados substituindo-se um caractere especial, no caso, o ponto de interrogação. Essa abordagem torna as consultas e atualizações mais rápidas, já que o SGBD apenas as executará, economizando o tempo de compilação.

Com a comodidade fornecida pelo uso do Axis2, é necessário apontar apenas uma classe Java no momento da criação do serviço. A classe criada no projeto chama-se *DoarSangueWebService* e contém todos os métodos que serão detalhados pelo WSDL e enviados em mensagens SOAP. Esses métodos utilizam os DAOs para acessar o banco de dados e retornam a resposta necessária para utilização nos clientes.

A Figura 5.3 ilustra o *Web Service* do Doador Sangue Bom e suas aplicações clientes.

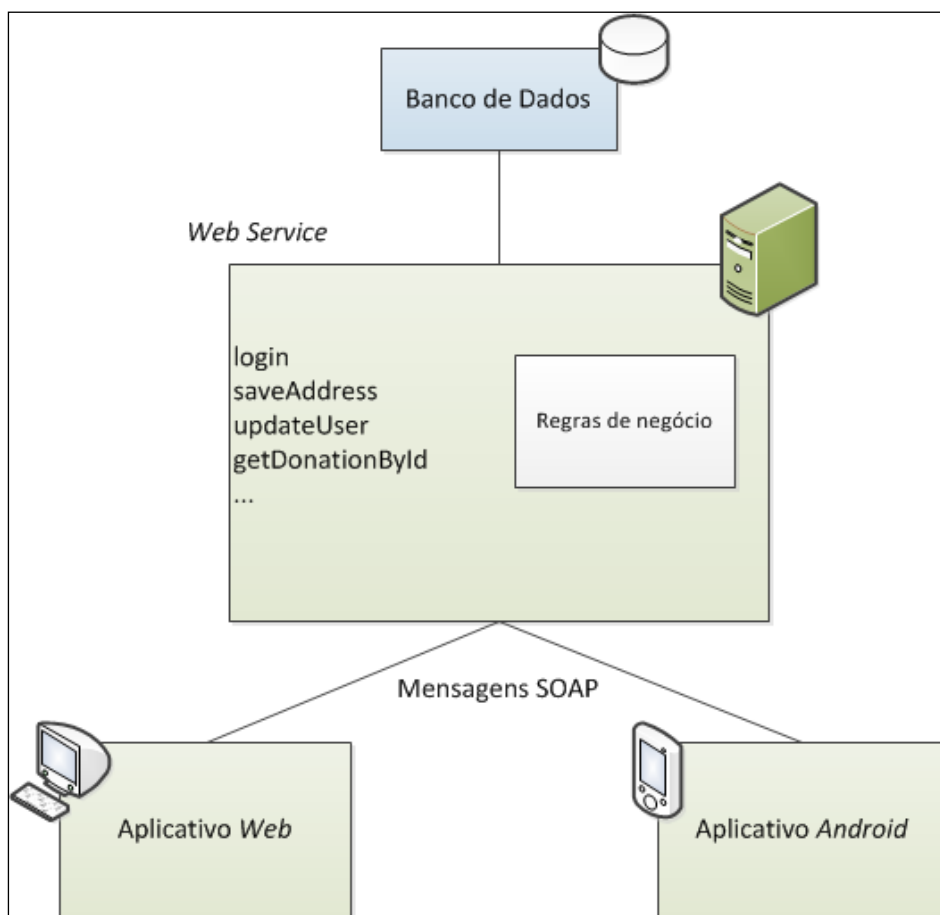


Figura 5.2. Web Service e clientes

5.3. Os métodos do Doador Sangue Bom

Ao longo do desenvolvimento do projeto foram escritos diversos métodos que realizam as funções idealizadas para o Doador Sangue Bom. A maioria dos métodos é utilizada em ambas as aplicações clientes, embora alguns, como os destinados a coletores e administradores, só são utilizados na aplicação *web*.

A Tabela 5.2 mostra os métodos do Doador Sangue Bom em ordem alfabética. Essa tabela pode ser chamada de tabela DOUA, já que possui todas as informações

necessárias para descrever o sistema: seus métodos e parâmetros, sua funcionalidade e os atores que estão autorizados a acessar cada um dos métodos.

Nome do método	Parâmetros	O que faz?	Atores autorizados
<i>checkCollectorCode</i>	Um código de coletor.	Checa se um código de coletor é válido.	Doador/Receptor, Coletor, Administrador
<i>checkExistingEmail</i>	E-mail.	Verifica se já existe um e-mail cadastrado.	Não se aplica
<i>checkUserAddress</i>	Um usuário.	Checa se um usuário já possui endereço cadastrado.	Doador/Receptor, Coletor, Administrador
<i>countDonationDisposal</i>	Uma doação.	Retorna a quantidade de intenções para uma determinada doação.	Doador/Receptor, Coletor, Administrador
<i>createCollector</i>	Um hospital e um código de coletor.	Cadastra um novo coletor de acordo com um código de coletor.	Administrador
<i>createDonationDisposal</i>	Um usuário e uma doação.	Salva a intenção de doação de um usuário.	Doador/Receptor, Coletor, Administrador
<i>createNews</i>	Título e texto de uma nova notícia.	Insere uma notícia no sistema.	Administrador
<i>deleteDonationDisposal</i>	Uma intenção de doação.	Remove uma intenção de doação.	Doador/Receptor, Coletor, Administrador
<i>deleteHistoryLog</i>	Uma entrada de histórico.	Remove uma entrada de histórico.	Doador/Receptor, Coletor, Administrador
<i>deleteHospital</i>	Um hospital.	Remove um hospital.	Administrador
<i>deleteUser</i>	Identificador do usuário.	Exclui um usuário.	Administrador
<i>getAllDonations</i>	Não se aplica.	Retorna todas as doações cadastradas.	Doador/Receptor, Coletor, Administrador
<i>getAllHospitals.</i>	Não se aplica.	Retorna todos os hospitais do sistema.	Doador/Receptor, Coletor, Administrador
<i>getAllUsers</i>	Não se aplica.	Retorna todos os usuários do sistema.	Administrador

<i>getBloodTypeStatistics</i>	Não se aplica.	Retorna as estatísticas de tipo sanguíneo.	Administrador
<i>getDonationById</i>	Identificador de uma doação.	Retorna as informações de uma doação específica.	Doador/Receptor, Coletor, Administrador
<i>getDonationDisposalsByHospital</i>	Um hospital.	Retorna todas as intenções de doação de um determinado hospital.	Coletor
<i>getDonationsByBloodType</i>	Um tipo sanguíneo.	Retorna todas as doações de determinado tipo sanguíneo.	Doador/Receptor, Coletor, Administrador
<i>getDonationsByHospital</i>	Um hospital.	Retorna todas as doações de determinado hospital.	Doador/Receptor, Coletor, Administrador
<i>getGenderStatistics</i>	Não se aplica.	Retorna as estatísticas de gênero.	Administrador
<i>getHospitalInfoById</i>	Identificador de um hospital.	Retorna as informações de um hospital específico, como nome, endereço, CNES e telefone.	Doador/Receptor, Coletor, Administrador
<i>getStatesNames</i>	Não se aplica.	Retorna o nome de todos os estados do Brasil.	Não se aplica
<i>getUserHistory</i>	Um usuário.	Retorna o histórico de um usuário.	Doador/Receptor, Coletor, Administrador
<i>hasAuthorization</i>	Um usuário.	Verifica se um usuário tem autorização a um recurso do sistema.	Não se aplica
<i>insertHospital</i>	Nome, telefone e CNES de um hospital.	Insere um hospital no sistema.	Administrador
<i>insertPersonalHistoryLog</i>	O usuário que adicionou o histórico, o nome de um hospital, o nome de um paciente e a data de doação.	Salva uma nova entrada no histórico de um usuário.	Doador/Receptor, Coletor, Administrador
<i>insertUser</i>	Nome, sobrenome, e-mail, senha, gênero, tipo sanguíneo, data de nascimento, identificação de ator do sistema.	Adiciona um novo usuário ao sistema.	Não se aplica

<i>login</i>	E-mail e senha.	Autoriza um usuário no sistema.	Não se aplica
<i>readNews</i>	Não se aplica.	Retorna todas as notícias do sistema.	Doador/Receptor, Coletor, Administrador
<i>saveAddress</i>	Endereço, bairro, cidade e estado.	Salva o endereço de um usuário no banco de dados.	Doador/Receptor, Coletor, Administrador
<i>saveDonation</i>	Identificador do usuário que fez a requisição, nome e tipo sanguíneo do paciente, hospital, motivo e número de bolsas de sangue (opcional).	Cadastra uma nova doação no sistema.	Doador/Receptor, Coletor, Administrador
<i>updateCollectorUserId</i>	Um usuário e um código de coletor.	Define um usuário do sistema como coletor.	Não se aplica
<i>updateHospital</i>	Informações do hospital, como CNES, nome, telefone.	Atualiza um hospital.	Administrador
<i>updateHospitalAddress</i>	Informações do novo endereço e um hospital.	Atualiza o endereço de um hospital.	Administrador
<i>updateUser</i>	Nome, sobrenome, e-mail, senha, gênero, tipo sanguíneo, data de nascimento, idioma de escolha.	Atualiza os dados de um usuário.	Doador/Receptor, Coletor, Administrador
<i>updateUserAddress</i>	Informações do endereço e um usuário.	Atualiza o endereço de um usuário.	Doador/Receptor, Coletor, Administrador
<i>validateDisposal</i>	Uma intenção de doação.	Valida uma intenção de doação.	Coletor

Tabela 5.2. Métodos do Doador Sangue Bom

Capítulo 6 - Cliente I - Aplicação Web

Atualmente, um cliente *web* é praticamente essencial para serviços que utilizem a *Internet*. Com o crescimento da computação em nuvem, as pessoas começaram a utilizar largamente serviços disponibilizados na *web* e acessados via *browser*. Aplicações que antes estavam apenas disponíveis para *desktop*, já encontram-se disponíveis via *web*.

Para o Doador Sangue Bom, a aplicação *web* é essencial e engloba a principal forma de acesso ao sistema. Vale lembrar que administradores e coletores possuem uma interface específica na aplicação *web*, onde serão desempenhadas suas funções.

6.1. Tecnologias utilizadas

A aplicação *web* foi desenvolvida com o *framework* de desenvolvimento *web* *JavaServer Faces* 2.1 (JSF) e algumas bibliotecas como *PrimeFaces* 3.3 [29] para a elaboração da interface e *gmaps4jsf* 1.1.4 [30] para utilização dos serviços de mapas do Google.

O *PrimeFaces* é uma biblioteca de componentes para JSF que oferece mais de 100 componentes que conferem mais dinamismo à página *web*, enriquecendo a interface com o usuário, apresentando design agradável, eficiência e flexibilidade para desenvolver. Trata-se de uma biblioteca com documentação rica, ampla participação da comunidade em fóruns e de rápido aprendizado das funcionalidades de seus componentes. Componentes são recursos visuais como calendário, gráficos, editor de texto, autocompletar, entre outros. Vale lembrar que o *PrimeFaces* também oferece suporte a Ajax, um conjunto de tecnologias que conferem mais dinamismo às páginas *web*.

Apesar de existirem outras opções de bibliotecas de componentes JSF, como *ICEFaces* e *RichFaces*, optamos pelo uso do *PrimeFaces* por ser a mais nova entre essas bibliotecas e por estar em constante ascensão [31].

O *gmaps4jsf* é uma biblioteca de código aberto que simplesmente integra a API do Google Maps com os recursos oferecidos pelo JSF, facilitando o uso de mapas em um sistema, permitindo a visualização, adição de marcadores, visão de ruas, além de diversos controles que enriquecem o uso de mapas em um sistema.

A Tabela 6.1 descreve cada uma das ferramentas utilizadas para desenvolver a aplicação *Web*.

Ferramenta	Explicação
Netbeans IDE 7.0.1	O IDE para desenvolvimento da aplicação <i>web</i> .
Java SE 6 Development Kit (JDK)	Kit de desenvolvimento para escrever aplicações em Java.
Apache Tomcat 7.0.14	Servidor web da aplicação.
<i>PrimeFaces</i> 3.3	Biblioteca para elaboração da interface da aplicação.
<i>gmaps4jsf</i> 1.1.4	Biblioteca de código aberto que integra os elementos do JSF com a API do Google Maps.

Tabela 6.1. Ferramentas utilizadas para desenvolvimento da aplicação web

6.2. *JavaServer Faces* (JSF)

O *JavaServer Faces* é um *framework* para desenvolvimento de aplicações *web*, baseado em Java. Ele permite a construção de interfaces de usuário com o uso de componentes. Além de representar componentes, a API do JSF permite gerenciar seus estados, manipular eventos, conversão de dados, validação de dados *server-side* (no lado do servidor), dá suporte à internacionalização e permite definir a navegação entre páginas [32].

Uma das maiores vantagens do JSF é a clara separação entre as regras de negócio e a apresentação. Isso significa que antes de se comunicarem entre si, a camada de lógica pode ser implementada separadamente da camada de visualização. Essa arquitetura permite um trabalho em conjunto mesmo em uma equipe multidisciplinar. Um ou mais *designers* poderiam ficar responsáveis pela camada de apresentação juntamente com codificadores HTML, enquanto um ou mais desenvolvedores ficariam responsáveis pela camada de lógica do sistema [33]. A camada de apresentação é representada pelos arquivos XHTML e a camada de lógica é representada por classes Java chamadas *Managed Beans*.

No caso do Doador Sangue Bom, as regras de negócio estão presentes no *web service* e fornecem o acesso aos dados. No lado da aplicação *web*, a classe *WSAccess* fornece acesso às funções do *web service* e repassa os dados recebidos para os *Managed Beans*, responsáveis por tratar esses dados e fornecê-los para as páginas XHTML. Tal processo é elucidado na figura 6.1.

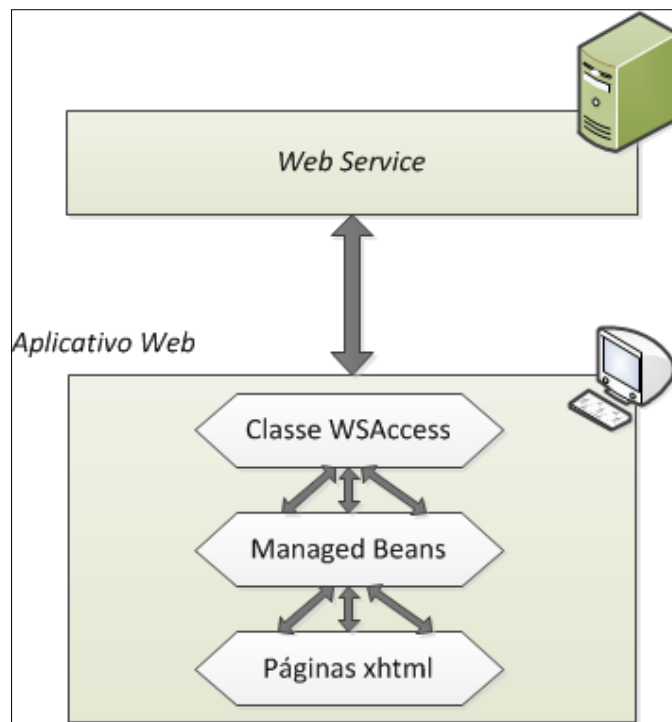


Figura 6.1. Separação de camadas no aplicativo web

6.2.1. Ciclo de Vida

Quando uma requisição JSF é realizada, seu processamento se dá em diversos passos. Essas etapas compõem o ciclo de vida JSF e estão divididas em seis fases. São elas: *Restore View*, *Apply Request Values*, *Process Validation*, *Update Model Values*, *Invoke Application* e *Render Response* [34][35]. Essas etapas podem ser vistas na Figura 6.2.

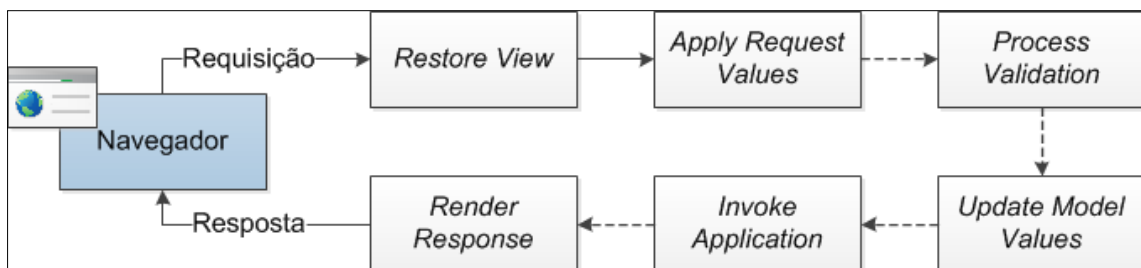


Figura 6.2. Ciclo de Vida JSF

6.2.1.1. *Restore View*

Na primeira fase, o servidor recebe a requisição e o JSF verifica qual página exibir de acordo com o nome presente na URL. Se a página requisitada ainda não estiver carregada em memória, o JSF irá carregá-la. A página é formada por um

conjunto de componentes, que são carregados em memória em estrutura de árvore, armazenada dentro de um objeto da classe *FacesContext*.

Se o usuário retornar a uma página já acessada, ação definida como *postback*, a página já estará carregada em memória. Cabe ao JSF apenas restaurar a árvore de componentes.

6.2.1.2. *Apply Request Values*

Na segunda fase, os componentes presentes na árvore de componentes são preenchidos com os parâmetros passados na requisição. Esses parâmetros também podem ser passados por cabeçalhos ou *cookies*.

6.2.1.3. *Process Validation*

É após a fase *Apply Request Values* que o primeiro tratamento de eventos ocorre, o que é representado na Figura 6.2 pelas setas pontilhadas e será tratado em detalhes na seção 6.1.1.7.

Na terceira fase, os valores dos componentes são validados de acordo com as regras definidas no desenvolvimento da aplicação.

Feita a comparação entre os valores e as regras de validação, verifica-se se algum valor é inválido. Se for, uma mensagem de erro é adicionada ao *FacesContext* e o JSF encaminha o processamento para a última fase, chamada *Render Response*, em que a página é reexibida com a mensagem de erro.

Caso não ocorra nenhum erro de validação, o JSF segue para a fase *Update Model Values*.

6.2.1.4. *Update Model Values*

Na quarta fase, são atualizados os valores do modelo no lado do servidor. Isso significa que, após a validação realizada na fase anterior, os valores dos componentes serão atribuídos às respectivas propriedades do *Managed Bean*.

Nesta fase também são realizadas as conversões de valor, seja por conversores nativos do Java ou por conversores implementados pelo desenvolvedor. Caso ocorra algum erro de conversão, uma mensagem de erro é adicionada ao *FacesContext* e, assim como na fase anterior, o JSF encaminha o processamento para a fase *Render Response*, e a página é reexibida com a mensagem de erro.

6.2.1.5. *Invoke Application*

Após realizadas as devidas validações, conversões e atribuições das propriedades do *Managed Bean*, na quinta fase os valores inseridos nos componentes poderão ser manipulados segundo a lógica de negócios criada na aplicação.

Considerando o caso de uso do sistema Doador Sangue Bom em que o usuário deseja requisitar uma doação, completado corretamente o formulário e pressionado o botão “Requisitar”, é na fase *Invoke Application* que o método responsável por inserir um pedido de doação no sistema é executado.

Nesta fase também é especificada a página de destino (*outcome*) a ser exibida após a execução do método requisitado.

6.2.1.6. *Render Response*

Na sexta e última fase, a página é construída e exibida com todos seus componentes contendo seus respectivos comportamentos e propriedades.

6.2.1.7. *Process Events*

Entre algumas das fases, ocorre um processo intermediário chamado *Process Events*. Ele é representado na Figura 6.2 pelas setas pontilhadas. Sua função é executar os eventos registrados e definir se o JSF deve seguir para a próxima fase ou avançar para a última fase (*Render Response*) para reexibir a página.

O direcionamento para a fase *Render Response* é necessário quando uma etapa sinaliza um erro de validação ou um erro de conversão, por exemplo. Se as ações determinadas pelos eventos se derem sem impedimentos, o JSF segue para a fase seguinte.

6.3. Internacionalização do Sistema

É de suma importância que a linguagem de um sistema seja compreendida pelo maior número possível de pessoas para que seu alcance não seja apenas local. Por isso, apesar de o Doador Sangue Bom ter sido desenvolvido com o português como idioma padrão, há a possibilidade de utilizar o sistema em inglês, para demonstrar que é possível traduzi-lo facilmente para qualquer idioma.

Todos os textos e mensagens encontram-se em um único arquivo de propriedades (*properties*) intitulado com o idioma que representa. O arquivo com os textos e mensagens em português tem o título *messages_pt_BR.properties* e o arquivo com textos e mensagens em inglês tem o título *messages_en_US.properties*. Esse conceito é conhecido como *locale* e consiste em definir a língua do usuário final, com o identificador da língua em minúsculo (*pt* para português, *en* para inglês, *es* para espanhol, etc.) e o identificador de região em maiúsculo (BR para Brasil, US para Estados Unidos, ES para Espanha, etc.).

Tais arquivos *properties* são compostos de um conjunto de chaves e valores. Todas as mensagens e textos do sistema se referem a essas chaves, que por sua vez determinam o valor (*string*) a ser exibido, dependendo da língua selecionada pelo usuário. Assim, quando houver necessidade de dar suporte a outro idioma, basta traduzir as *strings* de um arquivo, tarefa certamente oferecida por um profissional de línguas/tradução.

Trata-se de um método eficiente para a tradução de sistemas, já que as mensagens e textos estão centralizadas em um arquivo ao invés de estarem espalhados nos diversos arquivos que compõem o sistema.

No arquivo de configurações *faces-config.xml*, parte do projeto Java criado na IDE, são definidos os *locales* suportados pelo sistema e o *locale* padrão. Como mostra a Figura 6.3, o idioma padrão e o português do Brasil e o inglês dos Estados Unidos é suportado.

```
<application>
  <locale-config>
    <default-locale>pt_BR</default-locale>
    <supported-locale>en_US</supported-locale>
    <supported-locale>pt_BR</supported-locale>
  </locale-config>
  .
  .
  .
</application>
```

Figura 6.3. Idiomas do Doador Sangue Bom

O idioma exibido quando o usuário realiza *login* pela primeira vez é o mesmo de seu sistema operacional (caso o mesmo seja português do Brasil ou inglês dos Estados Unidos). O *locale* padrão (*default-locale*) é carregado quando o *locale* atual não for encontrado. É possível trocar o idioma de exibição do sistema em sua área de edição de conta (seção 6.4.1.5).

6.4. O *Website* do Doador Sangue Bom

Esta seção explica a navegação e as funcionalidades do *website* do Doador Sangue Bom. As funcionalidades e interfaces dependem do tipo de usuário que acessa o sistema, por isso, o *website* pode ser dividido em três categorias: interface do administrador, interface do coletor e interface do doador/receptor. O administrador e o coletor têm funcionalidades exclusivas que serão discutidas adiante, mas todos os tipos de usuários têm acesso às interfaces de doador/receptor.

6.4.1. Interface Doador/Receptor

Esta interface está disponível para todos os tipos de usuários e apresenta todas as funções principais da aplicação *web*.

6.4.1.1. Cadastro de Usuário (*cadastroUsuario.xhtml*)

A página de cadastro de usuário possui os campos necessários para o usuário se tornar parte do Doador Sangue Bom. São eles: nome, sobrenome, e-mail, gênero, tipo sanguíneo, data de nascimento e senha. Esses dados são validados para que não haja campos vazios, um endereço de e-mail inválido ou uma senha com menos de seis caracteres.

Após o preenchimento dos campos, deve-se clicar em “Salvar”. Se o formulário for validado, o usuário é criado e redirecionado para a página principal do sistema.

6.4.1.2. Página Principal (*doarSangue.xhtml*)

A página principal possui o menu principal, presente em todas as páginas. Ele dá acesso às páginas Doe (*doe.xhtml*), Receba (*requisitarDoacao.xhtml*), Alterar Cadastro (*editarUsuario.xhtml*), Histórico de Doações (*historicoDoacoes.xhtml*), Hospitais (*hospitais.xhtml*) e Informações (*informacao.xhtml*), além da opção Sair, para realizar *logout*.

A página também exibe as doações requisitadas no sistema, organizadas na seguinte ordem de exibição: doações do mesmo tipo sanguíneo do usuário e doações de outros tipos sanguíneos. Para visualizar uma doação o usuário deve clicar no botão “Doe”. Ele será redirecionado para a página de *visualizarDoacao.xhtml*, explicada na seção 6.4.1.9.

A Figura 6.4 mostra a página principal do sistema.



Figura 6.4. Página principal

6.4.1.3. Doe (doe.xhtml)

A página *doe.xhtml* apresenta ao usuário interessado em doar sangue os locais (hospitais) onde são requisitadas doações. Através de um mapa, disponibilizado pelo serviço do *Google Maps* com o uso da biblioteca *gmaps4jsf*, é possível visualizar esses locais.

Para incentivar essas doações, existe a opção de visualizar no mapa onde há requisições de doações perto do endereço do usuário. Este endereço pode ser cadastrado na página de Alteração de Cadastro (*editarUsuario.xhtml*), como será explicado na seção 6.4.1.5, ou definido através dos serviços de localização do Android, caso o usuário também utilize o aplicativo do Doador Sangue Bom.

Também existe a opção de visualizar todos os locais de requisição de doações, seja para ter uma noção da distribuição de pedidos ou para visualizar em quais hospitais há pedidos de doação. Essa funcionalidade é mostrada na Figura 6.5.



Figura 6.5. Mapa de todos os locais de doação

6.4.1.4. Receba (*requisitarDoacao.xhtml*)

A página *requisitarDoacao.xhtml* consiste em um formulário que deve ser preenchido por um usuário que deseja pedir doação para um familiar ou amigo.

O usuário deve inserir o nome do paciente que precisa de sangue, o tipo sanguíneo do paciente, o hospital em que o paciente está internado e o motivo pelo qual ele necessita de sangue. Depois basta clicar em “Requisitar” para que a requisição entre no sistema.

Ao inserir o hospital, o usuário interage com um recurso de autocompletar, que exibe sugestões do nome do hospital à medida que o usuário digita. Apenas hospitais cadastrados por um administrador são mostrados no autocompletar. O cadastro de hospitais é realizado por um administrador e é explicado na seção 6.4.3.2.

6.4.1.5. Alterar Cadastro (*editarUsuario.xhtml*)

Esta página tem como função a edição dos dados inseridos no cadastro (*cadastroUsuario.xhtml*), o cadastro de um endereço e a escolha do idioma do sistema.

Caso queira trocar o nome, sobrenome, e-mail ou outro dado inserido durante o cadastro, basta modificá-lo e clicar em “Alterar”. Assim como no cadastro, esses dados são validados para que não haja campos vazios ou um endereço de e-mail inválido.

Há também a opção de definir um endereço inserindo o próprio endereço, bairro, cidade e estado. O idioma do sistema pode ser escolhido entre as duas opções disponíveis: português (Brasil) e inglês (Estados Unidos). Feito isso, basta clicar em “Alterar”.

Se o usuário quiser desfazer sua conta, deve-se clicar no botão “Excluir conta”. Uma mensagem de confirmação é exibida e ao clicar em “Excluir”, todos os dados daquele usuário serão excluídos do sistema.

6.4.1.6. Histórico de Doações (historicoDoacoes.xhtml)

O histórico de doações é uma tabela contendo as doações já realizadas pelo usuário, sejam doações por intermédio do sistema Doador Sangue Bom ou não. As doações são exibidas com a data de quando foram realizadas, o hospital onde ocorreram e o nome da pessoa a quem o sangue foi destinado.

Para incluir uma doação realizada sem o intermédio do sistema Doador Sangue Bom, o usuário deve clicar em “Adicionar Doação”. Será aberto um painel em que deverão ser preenchidos o local da doação, o nome da pessoa a quem foi destinada a doação (caso tenha sido uma doação anônima, basta marcar a *checkbox* e ficará indicado como doação para a comunidade) e a data em que foi realizada. Ao clicar em “Adicionar”, a doação é inserida no histórico.

Para excluir uma doação do histórico é necessário selecionar uma doação da tabela e clicar em “Excluir Doação”. Uma mensagem de confirmação será exibida. Clicando em “Excluir”, a doação será excluída do histórico.

O histórico de doações é importante porque mantém um controle das doações já feitas no sistema, podendo ser utilizado por coletores para coletar estatísticas pertinentes a determinado hospital, além de permitir que o próprio usuário tenha noção de suas doações.

6.4.1.7. Hospitais (hospitais.xhtml)

A página de hospitais possui a lista de todos hospitais cadastrados pelo(s) administrador(es). Cada um deles é exibido com seu nome, telefone, CNES, endereço e um mapa mostrando sua localização.

Ao clicar em “Ver página” o usuário é direcionado para a página específica do hospital, que além das informações anteriores, possui também os pedidos de doações requisitados para ele, caso houver.

6.4.1.8. Informações (informacao.xhtml)

Esta página contém um conjunto de informações sobre o processo de doação de sangue, como quem pode ser doador, quais são os requisitos básicos para ser doador ou o que pode impedir provisoriamente uma pessoa de doar sangue.

6.4.1.9. Visualizar Doação (visualizarDoacao.xhtml)

A página de visualizar doação, mostrada na Figura 6.6, apresenta as informações pertinentes a uma solicitação de doação: o nome do paciente, seu tipo sanguíneo caso se aplique, o motivo da necessidade de sangue e os dados do hospital onde ele se encontra internado. Também possui o número de pessoas que já demonstraram intenção de doar através do sistema e uma área com botões para compartilhamento em redes sociais (Twitter, Facebook e Google+), permitindo a divulgação de doações, com o objetivo de aumentar o número de doadores e de usuários do sistema.

Para demonstrar intenção de doar, o usuário deve clicar em “Quero doar”. Isso fará com que o contador de intenções seja incrementado, somando uma intenção à contagem anterior. O usuário só pode demonstrar intenção de doar para uma dada solicitação uma vez. Por esse motivo, após clicar em “Quero doar” e a intenção for contabilizada, o botão se modifica para “Não vou mais doar”. Caso o usuário clique nele, sua intenção de doar será retirada e o contador de intenções será decrementado, subtraindo uma intenção da contagem anterior.



Figura 6.6. Página de visualizar doação

6.4.2. Interface Coletor

Esta seção explica a parte do sistema destinada ao coletor. Ela possui uma página específica para cadastro e uma página de validar doações, para coletores autenticados.

6.4.2.1. Cadastro de Coletor (*cadastroColetor.xhtml*)

A página de cadastro de coletor possui os mesmos campos da página de cadastro de usuário.. Além deles, há um campo de código, em que o coletor deve inserir um código gerado e fornecido por um administrador. Esses dados são validados para que não haja campos vazios, endereço de e-mail inválido ou um código inválido.

Após o preenchimento dos campos, deve-se clicar em “Salvar”. Se o formulário for validado, o usuário é criado, associado com o código de coletor já associado a um hospital pelo administrador (veja mais na seção 6.4.3.4) e redirecionado para a página principal do sistema.

6.4.2.2. Validar Doações (*validarDoacao.xhtml*)

A página de validar doações apresenta uma relação de intenções de doação declaradas pelos usuários para requisições de doação naquele hospital onde o coletor trabalha. Cabe ao coletor apenas validar as doações, ou seja, confirmar que o usuário que declarou intenção de doar efetivamente doou para o paciente a quem se propôs a doar.

Essa página é mostrada na Figura 6.7.

Código	Doador	Paciente	Data da Intenção	Situação	Validar
10	Megan Fox	James Cameron	13/09/2012	OK	<input checked="" type="checkbox"/>
11	Sylvester Stallone	Tim Berners-Lee	13/09/2012	Pendente	<input type="checkbox"/>
12	Jessica Alba	Tim Berners-Lee	13/09/2012	Pendente	<input type="checkbox"/>
13	Tom Hanks	Tim Berners-Lee	13/09/2012	Pendente	<input type="checkbox"/>

Figura 6.7. Validação de doações

6.4.3. Interface Administrador

Esta seção explica a parte do sistema destinada ao administrador. Ela possui páginas específicas para administradores autenticados com as funções de administrar usuários, administrar hospitais, inserir notícias, gerar código de coletor e visualizar estatísticas.

6.4.3.1. Administração de Usuários (*adminUsuario.xhtml*)

Uma tabela com as informações de todos os usuários está disponível para o administrador. Ela contém o código, nome, sobrenome, e-mail, gênero, tipo sanguíneo e idade dos usuários. É possível procurar por usuários segundo o nome, sobrenome, e-mail e idade. Também é possível ordenar a tabela de acordo com o gênero, o tipo sanguíneo ou idade. O administrador escolhe se deseja visualizar dez, vinte e cinco, cinquenta ou cem usuários por página.

Se for registrado abuso ou uso indevido do sistema, o administrador pode selecionar na tabela o usuário que causa transtornos e em seguida clicar em “Excluir Usuário”. Uma mensagem de confirmação será exibida com todos os dados do usuário que será excluído. Ao clicar em “Excluir”, aquele usuário é banido do sistema e todos seus dados são excluídos do mesmo.

6.4.3.2. Administração de Hospitais (adminHospital.xhtml)

A página de administração de hospitais, mostrada na Figura 6.8, disponibiliza uma tabela com as informações de todos os hospitais cadastrados e oferece as funcionalidades de adicionar, editar e excluir hospitais.

Para inserir um hospital, é preciso clicar em “Inserir hospital” e preencher os campos nome, telefone, CNES, endereço, bairro, cidade e estado referentes ao hospital. Depois, clica-se em “Salvar” para armazenar o hospital no banco de dados.

Feito isso, o hospital cadastrado poderá ser visualizado na tabela. Se houver necessidade de editar os dados do hospital, basta selecioná-lo na tabela e clicar em “Editar hospital”. Seus dados serão exibidos dentro de campos editáveis. Feita a edição, basta clicar em “Editar” e os dados serão modificados.

Caso haja necessidade de excluir um hospital, é preciso selecioná-lo na tabela e clicar em “Excluir hospital”. Uma mensagem de confirmação será exibida com todos os dados do hospital a ser excluído. Ao clicar em “Excluir”, aquele hospital é retirado do sistema.



[Usuários](#)
[Hospitais](#)
[Notícias](#)
[Gerar Código Coletor](#)
[Estatísticas](#)
[Sair](#)

Administrar Hospitais

Procurar

1 10

Código	Nome	Endereço	Bairro	Cidade	Estado
1	Hemorio	Rua Frei Caneca, número 08	Centro	Rio de Janeiro	Rio de Janeiro
2	INCA Hospital do Cancer I	Praça Cruz Vermelha, número 23	Centro	Rio de Janeiro	Rio de Janeiro
3	Hospital Pasteur	Av. Amaro Cavalcanti, número 495	Meier	Rio de Janeiro	Rio de Janeiro
4	Hospital Rio Mar	Av. Candido Portinari, número 555	Barra da Tijuca	Rio de Janeiro	Rio de Janeiro
5	Hospital São Lucas	Trav. Frederico Pamplona, número 32	Copacabana	Rio de Janeiro	Rio de Janeiro
6	Hospital Quinta D'Or	Rua Almirante Baltazar, número 383	São Cristovão	Rio de Janeiro	Rio de Janeiro
7	Hospital Pro Cardíaco	Rua Dona Mariana, número 217	Botafogo	Rio de Janeiro	Rio de Janeiro
8	Hospital Universitário Clementino Fraga Filho	Rua Professor Rodolpho Paulo Rocco, número 255	Ilha do Fundão	Rio de Janeiro	Rio de Janeiro
9	Hospital de Clínicas de Jacarepaguá	Rua Bacairis, número 499	Taquara	Rio de Janeiro	Rio de Janeiro
10	Hospital Beneficência Potuguesa	Rua Santo Amaro, número 80	Glória	Rio de Janeiro	Rio de Janeiro

1 10

[+ Inserir hospital](#)
[✎ Editar hospital](#)
[✕ Excluir hospital](#)

Figura 6.8. Página de administração de hospitais

6.4.3.3. Notícias (cadastroNoticia.xhtml)

A área de notícias permite que o administrador insira notícias pertinentes ao tema de doação de sangue. A página possui um campo para o título da notícia e um campo de edição de texto, onde é possível modificar a formatação do texto, inserir imagens, etc. Ao clicar em “Salvar”, a notícia é registrada no banco de dados e poderá ser visualizada na página principal.

6.4.3.4. Gerar Código de Coletor (adminCodigoColetor.xhtml)

A página de geração de código de coletor possui dois campos: código e hospital.

O campo “código” armazena um código alfa-numérico de dez caracteres fornecido por um algoritmo de geração aleatória de uma sequência de caracteres. Cada vez que o administrador clica em “Gerar código”, um código aleatório é criado e exibido no campo.

O campo “hospital”, assim como em *requisitarDoacao.xhtml*, possui o recurso de autocompletar, que exibe sugestões de nomes de hospitais cadastrados à medida que o administrador digita.

Ao clicar em “Salvar”, é criada no banco uma instância para representar o coletor, contendo o seu código e o hospital em que ele trabalha. Essa instância será relacionada a um usuário quando o coletor fizer o seu cadastro e fornecer o código previamente gerado pelo administrador.

O uso de um código para os coletores é importante para confirmar sua identidade. Esse código é passado para ele por um administrador após entrar em contato com o hospital. A figura do coletor é de uma pessoa que constata que um doador de fato realizou uma doação e entrega a ele um comprovante de seu ato de solidariedade. A existência de um sistema como o Doador Sangue Bom representaria uma vantagem para o hospital ao qual o coletor está vinculado, pois atrairia mais doadores, aumentando a disponibilidade de bolsas de sangue e diminuindo a dependência do banco de sangue central.

6.4.3.5. Estatísticas (*estatisticas.xhtml*)

A página de estatísticas oferece ao administrador a proporção de homens e mulheres cadastrados na forma de um gráfico de barras. Também apresenta o número de usuários de cada um dos tipos sanguíneos por meio de um gráfico no formato de pizza.

Capítulo 7 - Cliente II - Aplicativo Android

O mercado de dispositivos móveis vem crescendo vertiginosamente nos últimos anos, ganhando espaço e gerando lucro tanto para empresas quanto para desenvolvedores. O mundo inteiro está conectado através deles e ferramentas e serviços que utilizam o sistema de dispositivos móveis surgem diariamente.

O termo “dispositivo móvel” é usado não apenas para aparelhos celulares e *smartphones*, mas também para *tablets*, computadores portáteis com tela sensível ao toque, que dominaram o mercado nos últimos anos.

A gama de opções disponível para o usuário final é vasta e existem dispositivos que executam com diversos sistemas operacionais, dentre os quais se destaca o Android, do Google, e o iOS, da Apple. O iOS é o sistema operacional dos aparelhos fabricados pela própria Apple como o iPhone, iPod *Touch* e iPad. Já o Android é o sistema nativo utilizado por empresas como Samsung, LG e HTC em seus dispositivos.

O sucesso dos dispositivos móveis deve-se a três fatores principais. Um deles é a capacidade de conexão à *Internet*. *Smartphones* e *tablets* são capazes de acessar uma rede de dados móveis, como o EDGE, 3G ou 4G, além de conectar-se à *Internet* via *Wi-fi*. Outra característica que alavancou o sucesso deles foi a criação de lojas de aplicativos. Cada sistema operacional costuma ter uma loja de aplicativos, onde os usuários podem comprar ou adquirir gratuitamente diversas opções de aplicativos que ampliam a funcionalidade de seus dispositivos, aproveitando-se de seus recursos como câmera, *Bluetooth* ou GPS, por exemplo. Essa característica é tão marcante que definiu um novo mercado altamente lucrativo [36]. Por último, a evolução de *hardware* dos dispositivos móveis permitiu que sua capacidade computacional se tornasse semelhante a de computadores. Isso possibilita que os aplicativos desenvolvidos para eles sejam cada vez mais complexos.

Com o fácil acesso à *Internet*, aplicativos móveis clientes de *Web Services* tornaram-se comuns. Além disso, aplicativos que utilizam serviços de geolocalização podem prover funcionalidades bastante úteis.

Dadas a portabilidade de um aplicativo móvel e a capacidade de utilizar serviços de geolocalização, a criação de um aplicativo móvel para o Doador Sangue Bom mostrou-se atraente.

7.1. A escolha do sistema operacional móvel

Há várias opções de sistemas operacionais para dispositivos móveis disponíveis, como já explicado anteriormente, sendo o Android e o iOS as duas opções que mais se destacam, já que dominam o mercado, tanto através da venda dos dispositivos móveis que executam seus sistemas, quanto através da venda de aplicativos em suas respectivas lojas.

A possibilidade de desenvolver dois aplicativos, um para iOS e um para Android, é atraente, mas certas limitações no sistema operacional da Apple, que serão explicadas a seguir, não possibilitaram o desenvolvimento de um aplicativo para iOS durante o desenvolvimento deste projeto.

Os dois sistemas operacionais em questão possuem um *kit* de desenvolvimento de software, uma grande comunidade de desenvolvedores, emuladores para testes dos aplicativos e uma loja *online* para a publicação dos mesmos.

Aplicativos para iOS são desenvolvidos utilizando-se a linguagem de programação *Objective-C*, uma variação da linguagem C. Embora o *kit* de desenvolvimento da Apple seja gratuito, é necessário registrar-se como desenvolvedor para poder publicar aplicativos. Esse registro custa uma anuidade de 99 dólares [37]. O registro garante direito ao *kit* de desenvolvimento para iOS, ao Xcode (IDE) e o direito de publicar aplicativos na *App Store*, a loja de aplicativos da Apple. Além disso, é necessário ter um ambiente de desenvolvimento executando alguma versão do Mac OS X, o sistema operacional dos computadores da Apple.

Por outro lado, aplicativos para Android são feitos na linguagem Java com o auxílio do *kit* de desenvolvimento fornecido gratuitamente pelo Google. Aplicativos para Android podem ser desenvolvidos em qualquer ambiente. Para a publicação do aplicativo no *Google Play* (loja de aplicativos), o desenvolvedor também deve ser registrado, pagando o valor de 25 dólares para o Google.

O conhecimento prévio da linguagem Java, a maior disponibilidade de dispositivos que executam Android e a maior facilidade para adquirir um *kit* de desenvolvimento e publicar um aplicativo foram fatores essenciais na decisão de desenvolver um aplicativo para Android para o Doador Sangue Bom.

7.2. Tecnologias utilizadas

Para o desenvolvimento do aplicativo Android foi utilizado o Eclipse como IDE, devido ao seu suporte para desenvolvimento de aplicativos para Android com o ADT, um *plugin* com várias funcionalidades que auxiliam no desenvolvimento, inclusive facilitando bastante a elaboração de uma interface gráfica por meio de componentes.

O aplicativo foi desenvolvido para a versão 4.0.3 do Android, suportada tanto em *smartphones* quanto em *tablets*. Além disso, as APIs do Google, incluídas no *kit*, também tiveram que ser utilizadas para o suporte aos serviços de geolocalização.

Para a conexão com o *Web Service* foi utilizado o kSOAP 2, uma biblioteca de código aberto, exclusiva para Java, para aplicações clientes de *Web Services* que utilizam o protocolo SOAP. O kSOAP 2 é otimizado para aplicativos móveis e muitas de suas funções são simplificadas para melhor aproveitamento dos recursos limitados dos mesmos.

A Tabela 7.1 lista todas as ferramentas utilizadas no desenvolvimento do aplicativo para Android.

Ferramenta	Explicação
Eclipse Indigo	IDE para desenvolvimento.
Java SE 6 Development Kit (JDK)	Kit de desenvolvimento para desenvolver aplicações em Java.
Android SDK Tools	Ferramentas de desenvolvimento para o sistema operacional.
Android 4.0.3 com Google APIs	Versão do sistema operacional utilizada para desenvolvimento. Essa versão dá suporte tanto para <i>smartphones</i> , quanto para <i>tablets</i> . As Google APIs dão suporte para utilização de serviços de geolocalização.
kSOAP 2 (2.4)	Biblioteca para acessar <i>Web Services</i> desenvolvidos com SOAP.

Tabela 7.1. Ferramentas utilizadas para desenvolvimento do aplicativo Android

7.3. Considerações técnicas

Esta seção do trabalho foca em explicar detalhes técnicos de extrema importância no aplicativo do Doador Sangue Bom: o uso dos serviços de geolocalização e a classe *AsyncTask*, necessária para operações que acessam a *Internet* ou que demandam maior processamento.

7.3.1. Geolocalização

Geolocalização refere-se à identificação da localização onde um usuário de algum dispositivo se encontra em determinado momento. Uma boa definição de geolocalização é dada pelo grupo Educause e diz que geolocalização é a prática que associa um recurso digital a uma localização física, utilizando latitude, longitude e altitude [38].

A coleta de dados necessária para definir uma localização é feita por um endereço de IP, triangulação ou GPS. As duas últimas formas são as mais utilizadas para definir a localização de dispositivos móveis, já que estes normalmente estão conectados a alguma rede e, atualmente, a maioria possui GPS. A definição de uma localização via IP costuma ser bastante imprecisa e não é muito utilizada [39].

Um usuário pode utilizar a rede de seu dispositivo (seja a rede de dados de uma operadora ou alguma rede *Wi-fi*) para definir sua localização através de triangulação. Esse processo consiste em definir um ponto através de outros pontos conhecidos, no caso, antenas de operadoras de telefonia móvel ou roteadores de redes *Wi-fi*. Esta forma de obter uma localização não requer nenhuma tecnologia adicional e um dispositivo com conexão à *Internet* é capaz de obter uma localização dessa forma. No entanto, sua precisão é menor se comparada com a utilização de um GPS. Atualmente, grande parte dos dispositivos no mercado podem se aproveitar das duas formas de aquisição de dados para refinar o seu resultado, uma combinação chamada de A-GPS. Em seu *kit* de desenvolvimento, o Google dá suporte para que o desenvolvedor utilize qualquer uma das formas na criação de aplicativos para Android.

A geolocalização possui diversas vantagens para os usuários de dispositivos móveis, podendo auxiliar usuários no trânsito, facilitar a busca de locais ainda desconhecidos, cadastrar facilmente endereços ou permitir o compartilhamento de localizações com outras pessoas.

O uso da geolocalização no aplicativo do Doador Sangue Bom tem duas utilidades: registrar o endereço de um usuário que não possua um cadastrado; pegar a localização atual dele para encontrar hospitais mais próximos. Vale lembrar que nenhuma das funções de geolocalização no aplicativo é obrigatória e tais serviços apenas são utilizados quando autorizados pelo usuário. Por questões de privacidade, não é aconselhável utilizar dados de localização dos usuários sem o consentimento dos mesmos.

A programação da geolocalização no Android ocorre da seguinte forma: instancia-se um objeto *LocationManager*, que retornará o serviço de localização e o deixará pronto para ser utilizado. Em seguida, cria-se um *LocationListener*, um objeto

que ficará sempre verificando possíveis mudanças na localização. Para fins de economia de bateria do dispositivo e menor tráfego de dados pela rede, é possível definir a precisão ou o tempo em que a localização será atualizada pelo sistema. Por exemplo, pode-se definir que em um raio de cem metros a localização é considerada a mesma.

Quando a localização é atualizada, o sistema retorna um par de coordenadas (latitude e longitude), que precisam ser interpretadas e convertidas em um endereço inteligível para o usuário final. Para isso, é necessária a utilização de uma instância da classe *Geocoder*. Com os métodos disponibilizados através dessa classe, é possível retornar o endereço de um local com informações como bairro, cidade, estado e país. Essa é a informação que será utilizada, finalmente, no aplicativo.

Ao final de um processo que utiliza geolocalização, é importante remover o objeto *LocationListener*, para que este não fique consumindo a banda e a bateria do dispositivo desnecessariamente.

É importante ressaltar que, como o aplicativo foi testado em um emulador, foi necessário simular localizações. Para isso, o desenvolvedor fornece coordenadas para o emulador, que as interpreta como uma mudança de localização. Tal simulação é feita através do DDMS, uma ferramenta de depuração para Android, que permite simular a inserção de dados não disponíveis para o emulador como coordenadas ou números de telefone. Também é possível simular o tipo de rede ou o envio de mensagens entre dispositivos.

Esta seção explicou como funciona a geolocalização no Android. Detalhes sobre as funcionalidades que a utilizam no aplicativo do Doador Sangue Bom serão dados na seção 7.4.

7.3.2. *AsyncTask*

Primeiramente, é importante explicar o conceito de *threads*. *Threads* são fluxos de controle dentro de programas. A execução de um programa tem um início, um meio e um fim, assim como uma *thread*. No entanto, uma *thread* é executada paralelamente ao programa, sem obedecer à sequência que o define. Analogamente, uma *thread* pode ser vista como um programa executando dentro de outro programa, embora ela não possa ser considerada um programa, já que não pode ser executada separadamente.

Threads são normalmente utilizadas para operações que demandam maior processamento ou tempo, impedindo que um programa fique travado enquanto a

operação é feita. Vale notar também que os recursos utilizados dentro de uma *thread* não podem ser acessados fora dela.

Quando um aplicativo Android é iniciado, o sistema cria uma *thread* principal de execução para ele. É nessa *thread* que o aplicativo lida com a renderização da parte gráfica do aplicativo, além de detectar os eventos que ocorrem, como o toque na tela ou a seleção de um botão, por exemplo.

Todas as funções implementadas são executadas nessa *thread* principal e não são criadas novas *threads*, a não ser que o desenvolvedor o faça. Tarefas que demandam tempo para serem executadas podem fazer com que o aplicativo trave, já que ele terá de esperar até que sua execução seja concluída. Isso torna desaconselhável implementar esse tipo de tarefa na *thread* principal. Além disso, as funcionalidades que acessam a *Internet* são proibidas de executar na *thread* principal, gerando exceções e impedindo que o programa continue a ser executado. Essa característica obriga o acesso a *Web Services*, por exemplo, a ser realizado fora da *thread* principal.

Para lidar com esse problema de forma eficiente, o Google disponibiliza uma classe abstrata chamada *AsyncTask*. Ela permite que tarefas sejam executadas assincronamente e, após sua execução, seus resultados são enviados para a *thread* principal. Dessa forma, essas tarefas não influenciam a *thread* principal, deixando o sistema mais fluido e evitando travamentos [40].

Uma classe que estenda uma *AsyncTask* deve implementar quatro métodos:

- *onPreExecute* - O que deve ser executado antes. Faz todas as preparações necessárias para a execução da tarefa. Por exemplo, cria uma barra de progresso.
- *doInBackground* - A tarefa que será executada em *background*. Tarefas que acessam a rede ou que demandam maior processamento são feitas nesse método. Por exemplo, o acesso a um *web service*.
- *onProgressUpdate* - Define o que fazer durante a execução da tarefa. Por exemplo, atualiza a barra de progresso.
- *onPostExecute* - Define o que deve ser executado após o término da tarefa. Todas as atualizações na interface gráfica devem acontecer nesse método.

Como o Doador Sangue Bom acessa a *Internet* para conectar-se ao seu *Web Service*, foi necessário o uso de classes que estendam a classe *AsyncTask*. Por

exemplo, o *login* no sistema ou o cadastro de uma nova doação é feito em *background*.

7.4. O aplicativo do Doador Sangue Bom

Esta seção detalha o aplicativo do Doador Sangue Bom e suas funcionalidades.

O aplicativo foi dividido em onze atividades, que serão explicadas nas próximas seções. No desenvolvimento para Android, uma atividade consiste de uma tela para o usuário interagir. Uma atividade pode trocar dados com outras atividades, mas elas são independentes entre si.

Para a interface do aplicativo adotou-se um padrão de gestos, sempre que aplicável. Por exemplo, para visualizar hospitais ou doações, basta arrastar a tela para o lado com o dedo. Essa forma torna a visualização mais agradável e fácil para o usuário final, sendo análoga a aplicativos como um álbum de fotos, por exemplo. No Android, essa forma de visualização é obtida com um objeto *ViewFlipper*, que, ao inserido em uma atividade, permite que várias telas sejam alternadas com gestos ou botões.

7.4.1. *Login* e tela inicial

A tela de *login* do aplicativo é simples e pede que o usuário insira seu e-mail de cadastro e sua senha. Caso o usuário não seja cadastrado, também é possível se cadastrar direto do aplicativo. A Figura 7.1 mostra a tela de *login*.



Figura 7.1. Tela de login do aplicativo

Após o acesso, o usuário é direcionado para a tela inicial do sistema. Caso ele não possua um endereço cadastrado, o aplicativo perguntará se ele deseja cadastrar um endereço automaticamente, utilizando os serviços de localização, se o usuário estiver em casa. Com isso, não há a necessidade de digitar nenhuma informação. Por outro lado, o usuário também pode optar digitar seu endereço.

A tela inicial do sistema é simples e intuitiva, possuindo botões que levam para outras funcionalidades do sistema: visualização de doações, requisição, edição de informações, visualização de hospitais, notícias, histórico do usuário e troca de endereço.

7.4.2. Visualização de Doações

A tela de visualizar doações exibe o tipo sanguíneo do paciente, o endereço do hospital que aceita as doações e o motivo. Além disso, há três opções acessíveis a partir dessa tela. A primeira delas exibe informações detalhadas sobre a doação em questão, que será melhor explicada na seção 7.4.3. A segunda opção calcula o endereço do hospital mais próximo de onde o usuário mora e exibe uma doação que esteja sendo aceita no local. A terceira repete o processo anterior, mas com base na localização atual do usuário, ao invés de seu endereço. Essas funcionalidades utilizam-se dos serviços de localização do Google. Vale lembrar que essas

funcionalidades requerem o uso de dados, consumindo banda da *internet* móvel do usuário.

A Figura 7.2 mostra a tela de visualização de doações.

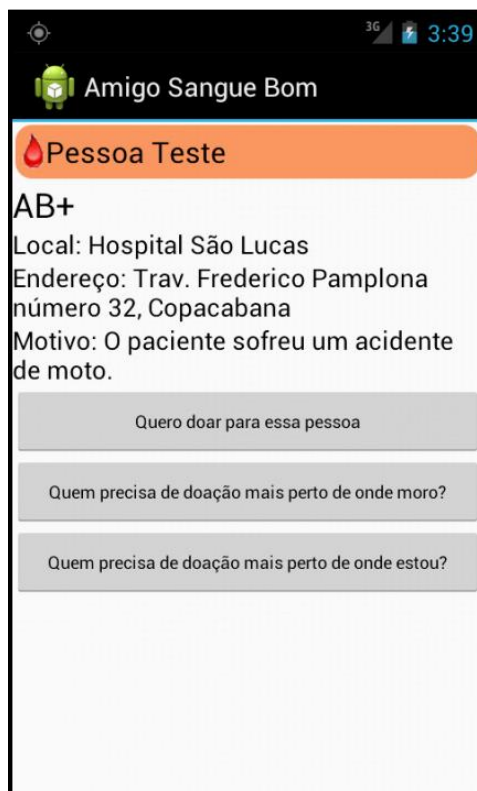


Figura 7.2. Tela de visualização de doações do aplicativo

7.4.3. Doação específica e mapa

A tela de doação específica é exibida quando um usuário escolhe a opção de doar para alguma pessoa na tela de doação. Nessa tela, todos os detalhes sobre aquela doação serão especificados e o usuário pode declarar a intenção de doar sangue para a pessoa escolhida. Essa intenção será validada posteriormente por um coletor.

Também há a opção de compartilhar aquele pedido em diversos aplicativos como Facebook ou Twitter, por exemplo, tornando possível o aumento de visibilidade das doações e do próprio sistema.

Também é possível visualizar o local de doação em um mapa, o que abrirá uma atividade especializada em exibição de mapas em aplicativos Android. O mapa é carregado dos servidores do Google e foi configurado com duas opções principais: mostrar um hospital mais próximo do endereço cadastrado do usuário ou mostrar um

hospital mais próximo de onde o usuário esteja naquele momento, através do uso dos serviços de localização.

Ícones são exibidos nos hospitais que necessitam de doação, assim como no endereço do usuário. O usuário pode aproximar ou afastar a visão do mapa para melhor visualização dos endereços. No botão de “Mais opções” há um opção extra que exibe todos os hospitais que estejam aceitando doações, como mostra a Figura 7.3. Mais opções extras podem ser implementadas futuramente.



Figura 7.3. Tela de mapa do aplicativo

7.4.4. Requisitar doação

Ao requisitar uma doação, o usuário deve preencher as mesmas informações que são necessárias na aplicação *web*.

O pedido requisitado é salvo no banco de dados e pode ser imediatamente visualizado por qualquer usuário do sistema, seja acessado pelo próprio aplicativo, como pela aplicação *web*.

7.4.5. Visualização de hospitais

A tela de visualização de hospitais segue o mesmo padrão das telas de visualização do sistema. Essa tela exibe as informações essenciais sobre cada hospital, como o nome, o telefone, o endereço e o CNES. Também é possível visualizar apenas as doações daquele hospital.

7.4.6. Notícias

Semelhante à tela de visualização de hospitais, a tela de notícias segue o mesmo padrão adotado no restante do sistema e exibe as notícias que foram cadastradas no sistema por um administrador.

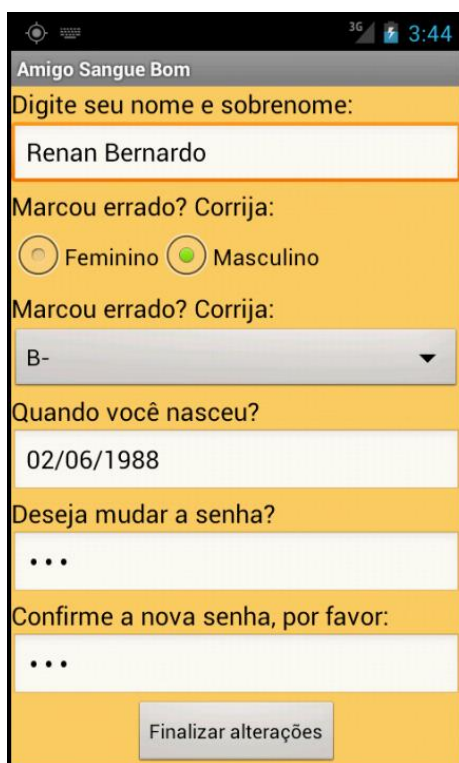
7.4.7. Cadastro e edição de usuário

A tela de cadastro de usuário pede exatamente as mesmas informações do cadastro feito pela aplicação *web*. Após feito um novo cadastro, o usuário conecta-se automaticamente no sistema.

Ao cadastrar um usuário, uma validação dos dados inseridos é feita para checar se todas as informações foram inseridas e, se sim, se estão no formato correto. Por exemplo, a data de nascimento inserida deve ser válida e a senha deve ser digitada duas vezes para evitar erros de digitação do usuário.

A tela de edição de usuário é muito semelhante à tela de cadastro e os dados também são validados ao serem editados.

A Figura 7.4 mostra a tela de edição de usuário.



A captura de tela mostra a interface de edição de usuário de um aplicativo. No topo, há uma barra de status com ícones de rede, bateria e o horário 3:44. Abaixo, o título da tela é "Amigo Sangue Bom". O formulário contém os seguintes campos e controles:

- Um campo de texto com o rótulo "Digite seu nome e sobrenome:" contendo o texto "Renan Bernardo".
- Dois botões de opção com o rótulo "Marcou errado? Corrija:"; o primeiro é "Feminino" (desativado) e o segundo é "Masculino" (ativado).
- Um menu suspenso com o rótulo "Marcou errado? Corrija:" mostrando a opção "B-".
- Um campo de texto com o rótulo "Quando você nasceu?" contendo a data "02/06/1988".
- Dois campos de senha com o rótulo "Deseja mudar a senha?". O primeiro campo contém pontos para ocultar o texto.
- Um campo de confirmação de senha com o rótulo "Confirme a nova senha, por favor:" contendo pontos para ocultar o texto.
- Um botão "Finalizar alterações" na base da tela.

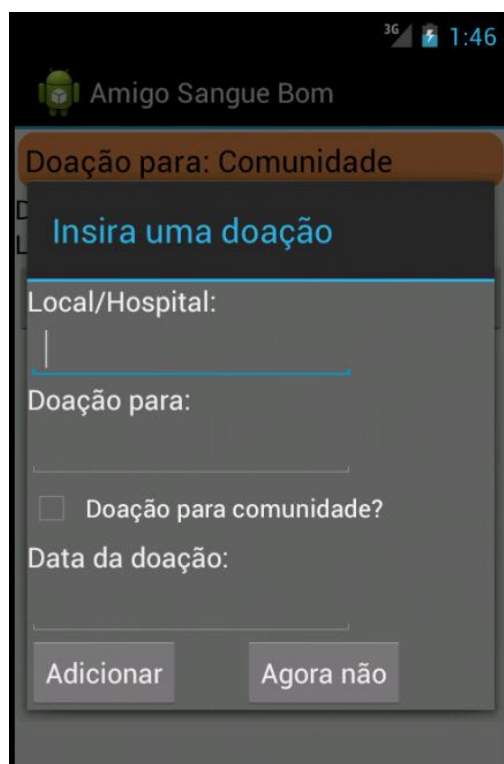
Figura 7.4. Tela de edição de usuário do aplicativo

7.4.8. Histórico

O histórico de doações do usuário também pode ser acessado no aplicativo e segue o mesmo padrão de interface adotado no restante do sistema, permitindo que as entradas do histórico sejam visualizadas através do gesto de arrastar a tela para o lado.

Também é possível adicionar doações que não foram feitas através do sistema, como mostra a Figura 7.5. Dessa forma, o usuário pode ter uma lista de todas as contribuições já feitas.

A importância do histórico está explicada na seção 6.4.1.6.



3G 1:46

Amigo Sangue Bom

Doação para: Comunidade

Insira uma doação

Local/Hospital:

Doação para:

☐ Doação para comunidade?

Data da doação:

Adicionar Agora não

Figura 7.5. Tela de inserção de doação em histórico no aplicativo

7.5. Publicação no *Google Play*

O *Google Play* é a loja de aplicativos para Android e possui um vasto acervo de aplicativos, pagos ou gratuitos. O aplicativo do Doador Sangue Bom foi publicado de forma gratuita e apenas com o objetivo de aprendizado, já que o projeto não estava em produção ao seu término.

Para a publicação de um aplicativo no *Google Play* é necessário atingir um número de requisitos. Caso algum deles não seja alcançado, o aplicativo ainda não está apto para ser publicado no *Google Play*.

Todo aplicativo para Android possui um arquivo XML denominado *AndroidManifest*, que possui informações necessárias para a execução e publicação de um aplicativo. Todas as atividades devem ser declaradas nesse arquivo, além das informações sobre quais recursos o aplicativo precisará de autorização para acessar, como o uso da internet, da câmera do dispositivo ou do *Bluetooth*, por exemplo.

A Figura 7.6 mostra o *AndroidManifest* do Doador Sangue Bom. Nela, é possível ver as autorizações necessárias para o aplicativo, como acesso à internet e uso dos serviços de localização. Uma característica importante desse arquivo no momento da publicação são os atributos *versionCode* e *versionName*. Ao publicar um aplicativo pela primeira vez, o atributo *versionCode* deve ser definido com um número e todas as atualizações do aplicativo devem incrementar esse valor. Já o atributo *versionName* segue um padrão de versionamento definido pelo desenvolvedor. Vale lembrar que é o *versionName* que será exibido para o usuário final. Os atributos *icon* e *label* da *tag application* também são importantes, já que definem o ícone do aplicativo e o nome pelo qual ele será reconhecido.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.DoarSangue.Android"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.SET_DEBUG_APP"></uses-permission>
    <uses-sdk android:minSdkVersion="11" android:targetSdkVersion="11" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <uses-library android:name="com.google.android.maps" />
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".donationsActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

Figura 7.6. AndroidManifest

Antes da publicação, também é importante checar se não há nenhuma função usada para testes que esteja exibindo informações desnecessárias para o usuário final, além de corrigir todos os erros, caso existam.

Verificados os requisitos supracitados, é necessário gerar uma versão de lançamento, que é um arquivo no formato APK (.apk). Um arquivo APK pode ser executado em qualquer aparelho com sistema operacional Android, desde que a versão do sistema esteja de acordo com a versão exigida pelo aplicativo. Esse arquivo precisa ser assinado com uma chave privada e otimizado para lançamento. O *kit* de desenvolvimento para Android e o Eclipse possuem ferramentas que auxiliam nesses dois processos.


A assinatura do aplicativo com um certificado é sempre exigida pela Google em seus aplicativos e serve como um modo de identificar o autor de um aplicativo, seja ele uma empresa ou pessoa física. A tarefa de assinar um aplicativo é bem simples, não necessitando de nenhuma autoridade de certificação, podendo ser feita diretamente pelo desenvolvedor do aplicativo. Caso o aplicativo não esteja assinado, nenhum dispositivo será capaz de executá-lo.


Com o aplicativo devidamente testado e assinado, é possível publicá-lo no Google Play. A Figura 7.7 mostra a tela exibida logo após o envio do aplicativo do Doador Sangue Bom. Ela mostra as informações que foram definidas no *AndroidManifest*, para que o autor possa conferir antes de concluir o processo de publicação.

Fazer upload do novo APK

Requerido: selecione o APK de seu aplicativo

com.DoarSangue.Android (194k)

Rascunho salvo 




Amigo Sangue Bom [\[Remover\]](#)

versionName: 1.0

versionCode: 1

Localizado para: padrão



Este apk solicita 6 permissões sobre as quais os usuários serão avisados


android.permission.INTERNET

android.permission.ACCESS_NETWORK_STATE

android.permission.ACCESS_FINE_LOCATION

android.permission.ACCESS_COARSE_LOCATION

android.permission.ACCESS_NETWORK_STATE



Este apk necessita de 5 recursos que serão usados para a filtragem do Google Play.

Opcional: adicionar um arquivo de expansão

Se seu aplicativo exceder o limite de APK de 50 MB, você pode adicionar arquivos de expansão. [Saiba mais](#)

Adicionar arquivo

Salvar

Figura 7.7. Publicação no Google Play

O aplicativo foi publicado em português e pode ser utilizado em aparelhos com o sistema operacional Android [41]. Atualizações podem ser facilmente lançadas no Google Play, bastando enviar uma versão mais recente do aplicativo.

A Figura 7.8 mostra a página do aplicativo no Google Play.

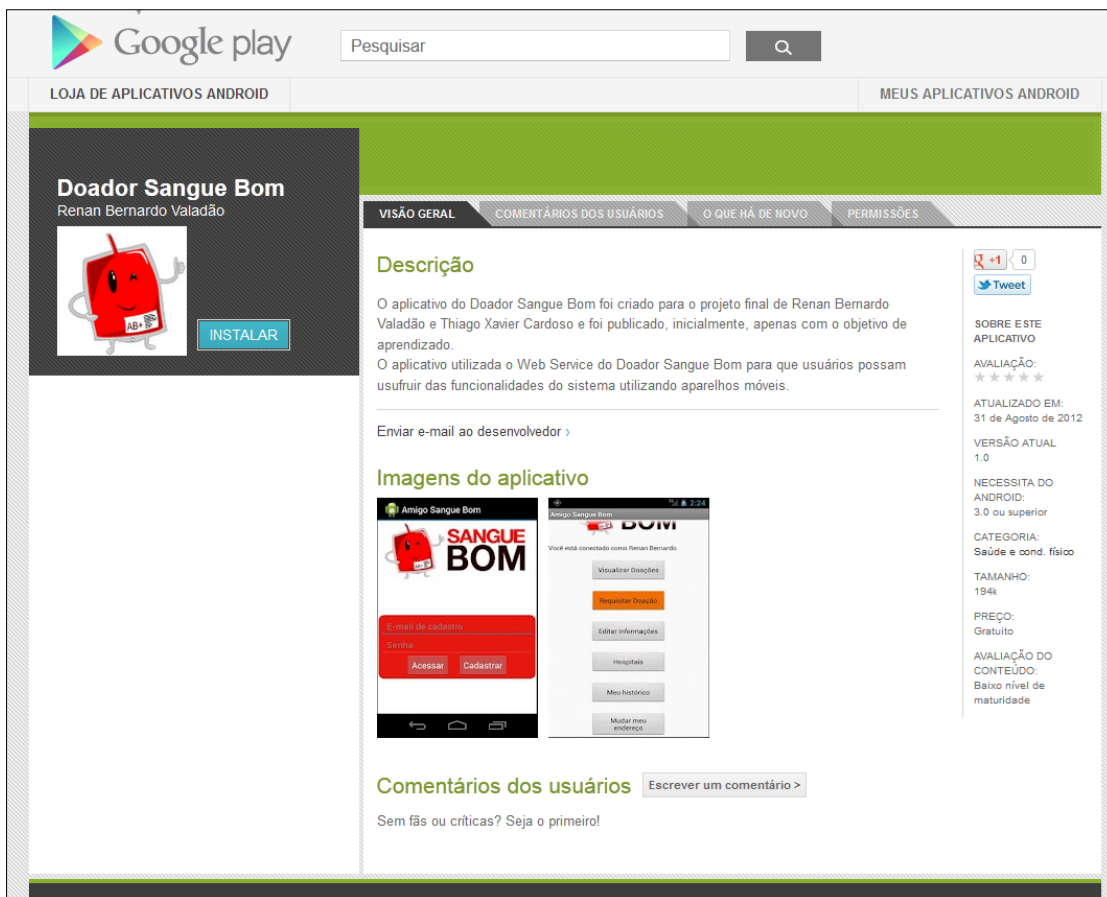


Figura 7.8. Publicação no Google Play II

Capítulo 8 – Conclusões

Este projeto final abordou diversos conceitos aprendidos durante a graduação, como modelagem de banco de dados, desenvolvimento de *Web Services*, aplicações *web* e aplicativos móveis, além de concentrar-se em um tipo de inovação social, tema abordado nas disciplinas de Empreendedorismo, Novos Modelos de Negócio e Inovação e Empreendedorismo.

Os membros do projeto tiveram a oportunidade de aprender diversos conceitos e tecnologias novas, além de elaborar um sistema que pode trazer grandes benefícios tanto para pacientes em necessidade de doações, assim como para estabelecimentos de saúde e até mesmo doadores regulares.

Com o uso de um *Web Service*, criou-se a possibilidade de outros desenvolvedores criarem aplicações que expandam o Doador Sangue Bom, trazendo novos benefícios para a sociedade na área da saúde.

Para o engenheiro em formação, esses aspectos são importantes, possibilitando o uso de diversas tecnologias novas simultaneamente, além de permitir uma visão de como é projetar um sistema com objetivos sociais, tornando possível estudar o impacto que tal sistema pode ocasionar na vida de seus usuários, além de permitir a ponderação de vantagens e desvantagens das tecnologias utilizadas no desenvolvimento.

Ciência e tecnologia são, sem dúvida, agentes transformadores de uma sociedade. O objetivo do engenheiro é utilizá-las como instrumentos para o progresso e o bem-estar de todos. Com vontade e competência é possível contribuir para que as pessoas tenham um futuro melhor.

Capítulo 9 - Trabalhos Futuros

Embora o Doador Sangue Bom esteja completamente funcional ao término deste trabalho, existem funcionalidades que podem ser implementadas ou expandidas futuramente, além da criação de novos clientes para consumir o *Web Service*.

9.1. Aplicativo para iOS

Com a utilização de uma arquitetura orientada a serviços, torna-se muito mais fácil a criação de novos clientes para o sistema, que só precisarão consumir o *Web Service* do sistema.

Dada a disponibilidade de um *kit* de desenvolvimento para iOS e um computador com o sistema operacional Mac OS X, seria interessante o desenvolvimento de um aplicativo para iOS, já que o mercado e a visibilidade dos aplicativos da Apple estão em constante expansão, além do número de pessoas que possuem dispositivos da Apple crescer constantemente. O aplicativo consumiria o *Web Service* do Doador Sangue Bom e teria funcionalidades bastante semelhantes ao do aplicativo Android.

9.2. Funcionalidades Além da Doação de Sangue

Embora o foco do sistema para este projeto tenha sido doações de sangue, é possível expandi-lo muito além disso.

Em uma versão futura do sistema, hospitais poderiam utilizá-lo para enviar resultados de exames de sangue para seus usuários, já que um exame é sempre feito em cada doação de sangue.

Outra funcionalidade que também poderia fazer parte do sistema, embora em uma área diferente, seria outros tipos de doação, que, em muitos hospitais, também sofre deficiência em sua divulgação e incentivo.

9.3. Modelo de Negócios

Apesar de ser um sistema que trabalha com ideias altruístas, há uma necessidade mínima de manutenção. Por isso, é preciso que o sistema seja capaz de gerar valor sem comprometer a experiência do usuário.

A opção de cobrar uma assinatura dos usuários foi totalmente descartada, visto que o objetivo é promover a doação de sangue e uma assinatura significaria um entrave para essa meta.

O uso de propagandas é uma alternativa, desde que seja utilizada com moderação. Para tal, seria integrado ao Doador Sangue Bom o *Google AdSense*, um sistema gratuito que permite aos administradores de um *website* gerarem receita exibindo anúncios promovidos pelo *Google*.

Também poderia haver uma parceria com os hospitais cadastrados. Cada hospital particular teria em sua página exclusiva dentro do sistema, além das informações básicas, uma área destinada a anúncios do próprio hospital ou de parceiros do mesmo.

A última opção seria aceitar doações dos usuários via *PayPal*. Aqueles que doassem dinheiro ao sistema seriam bonificados com pedidos de doação na primeira página e até mesmo pequenos brindes.

9.4. Nível de Urgência para Doações

Uma funcionalidade que pode se mostrar bastante útil para hospitais é definir um nível de urgência para doações. Muitos hospitais passam por problemas de falta de bolsas de sangue no estoque, o que pode deixar diversos pacientes em risco.

Com as vantagens e facilidades criadas com a utilização de serviços de localização e a exibição de mapas, seria possível mostrar o nível de urgência por região de um determinado local, permitindo que os usuários tenham ideia dos hospitais e regiões que mais necessitam de doações.

9.5. Hibernate

Embora tenha se optado por não utilizar o Hibernate, pode ser vantajoso utilizá-lo no futuro, caso a complexidade do modelo do banco de dados aumente.

9.6 Estatísticas Regionais de Doações

Talvez uma das mais importantes e ousadas funcionalidades seria obter estatísticas do número de doações, demanda por doações, número de pacientes com determinado tipo sanguíneo e estoque do banco de sangue dependendo da região.

Através dos endereços dos usuários e dos hospitais, seria possível dividir um município em áreas de doação e visualizar as mais bem atendidas e as mais

necessitadas. Essas informações poderiam contribuir para as autoridades locais promoverem campanhas de doação de sangue de acordo com as áreas delimitadas pelo sistema.

Com todas essas informações, os hospitais poderiam, também, controlar seus estoques de sangue.

Tais funcionalidades tornar-se-ão realidade apenas se o Doador Sangue Bom obtiver boa aceitação entre os usuários e os hospitais. A ideia é inicialmente oferecer o serviço para o município do Rio de Janeiro e expandir essa abrangência à medida que a aceitação for ampliando.

Referências

- [1] "Pesquisa sobre Doação de Sangue para Projeto Final", <https://docs.google.com/spreadsheet/viewform?formkey=dERNS1phVHJKa092ZkNITTdydVVnNGc6MQ>, acessado em 15 de setembro de 2012.
- [2] "Google Play", <https://play.google.com/>, acessado em 14 de setembro de 2012.
- [3] "Tortoise SVN", <http://tortoisesvn.tigris.org/>, acessado em 14 de setembro de 2012.
- [4] "Ministério da Saúde apresenta ações para ampliar doação de sangue", <http://portalsaude.saude.gov.br/portalsaude/noticia/5604/162/ms-apresenta-aco-es-para-ampliar-doacao-de-sangue.html>, acessado em 15 de setembro de 2012.
- [5] "Hemorio - Indicadores", http://www.hemorio.rj.gov.br/html/Apresentacoes/Indicadores_hemorio_2011/Dados.htm, acessado em 15 de setembro de 2012.
- [6] "Ministério alerta para redução no estoque de sangue", <http://portalsaude.saude.gov.br/portalsaude/noticia/5273/162/ministerio-alerta-para-reducao-no-estoque-de-sangue.html>, acessado em 15 de setembro de 2012.
- [7] "Estoque de sangue cai com chegada do Carnaval", <http://portalsaude.saude.gov.br/portalsaude/noticia/4238/162/estoque-de-sangue-cai-com-a-chegada-do-carnaval.html>, acessado em 15 de setembro de 2012.
- [8] MULGAN, G., TUCKER, S., ALI, R., SANDERS, B., *Social Innovation - What it is, why it matters and how it can be accelerated*, disponível em http://eureka.bodleian.ox.ac.uk/761/1/Social_Innovation.pdf, 2007, acessado em 15 de setembro de 2012.

- [9] “PostgreSQL vs. MySQL: How to Select the Right Open-Source Database”, 2010, acessado em 13 de setembro de 2012.
- [10] “CNESNet – Secretaria de Atenção à Saúde”, <http://cnes.datasus.gov.br>, acessado em 13 de setembro de 2012.
- [11] “Everware-CBDi”, <http://everware-cbdi.com/>, acessado em 13 de setembro de 2012.
- [12] WILKES, L., SPROTT, D., *Understanding Service-Oriented Architecture*, disponível em <http://msdn.microsoft.com/en-us/library/aa480021.aspx>, 2004, acessado em 13 de setembro de 2012.
- [13] HE, H., *What is Service-Oriented Architecture*, disponível em <http://www.xml.com/pub/a/ws/2003/09/30/soa.html>, 2003, acessado em 13 de setembro de 2012.
- [14] BAKSHI, A., *Service directory patterns employed in Java Web services as UDDI*, disponível em <http://www.techrepublic.com/article/service-directory-patterns-employed-in-java-web-services-as-uddi/5053805>, 2003, acessado em 13 de setembro de 2012.
- [15] “Web Services Architecture”, <http://www.w3.org/TR/ws-arch/>, 2004, acessado em 13 de setembro de 2012.
- [16] “Extensible Markup Language (XML)”, <http://www.w3.org/XML/>, acessado em 13 de setembro de 2012.
- [17] “Unicode in XML and other Markup Languages”, <http://www.w3.org/TR/unicode-xml/>, 2004, acessado em 13 de setembro de 2012.
- [18] SPIES, B., *Web Services, Part 1: SOAP vs. REST*, disponível em <http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest>, 2008, acessado em 13 de setembro de 2012.
- [19] “Web Services Description Language (WSDL) 1.1”, <http://www.w3.org/TR/wsdl>, 2001, acessado em 13 de setembro de 2012.

- [20] BELLWOOD, T., *Understanding UDDI*, disponível em <http://www.ibm.com/developerworks/webservices/library/ws-featuddi/>, 2002, acessado em 13 de setembro de 2012.
- [21] RODRIGUEZ, A., *RESTful Web Services: The basics*, disponível em <http://www.ibm.com/developerworks/webservices/library/ws-restful/>, 2008, acessado em 13 de setembro de 2012.
- [22] TYAGI, S., *RESTful Web Services*, disponível em <http://www.oracle.com/technetwork/articles/javase/index-137171.html>, 2006, acessado em 13 de setembro de 2012.
- [23] MORO, T. D., DORNELES, C. F., REBONATTO, M. T., *Web services WS-* versus Web Services REST*, disponível em http://www.upf.tc.br/computacao/images/stories/TCs/arquivos_20092/Tharcis_Dal_Moro.pdf, 2009, acessado em 13 de setembro de 2012.
- [24] ENGLANDER, R. *Java and SOAP*, 1 ed., O'Reilly Media, 2002.
- [25] "Web Services - Axis", <http://axis.apache.org/axis/>, acessado em 14 de setembro de 2012.
- [26] JAYASINGHE, D., *Looking into Axis2*, disponível em <http://careerride.com/Apache-Axis2-Architecture.aspx>, 2007, acessado em 13 de setembro de 2012.
- [27] "Apache Tomcat", <http://tomcat.apache.org/>, acessado em 14 de setembro de 2012.
- [28] "Core J2EE Patterns - Data Access Object", <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>, acessado em 13 de setembro de 2012.
- [29] "PrimeFaces - Ultimate JSF Component Suite", <http://www.primefaces.org/>, acessado em 14 de setembro de 2012;

- [30] “gmaps4jsf - The Complete Integration of Google Maps with JavaServer Faces”, <http://code.google.com/p/gmaps4jsf/>, acessado em 14 de setembro de 2012.
- [31] “Why Primefaces?”, <http://www.primefaces.org/whyprimefaces.html>, acessado em 13 de setembro de 2012.
- [32] “The Java EE5 Tutorial”, <http://docs.oracle.com/javaee/5/tutorial/doc/bnapi.html>, acessado em 13 de setembro de 2012.
- [33] PROHORENKO, A., PROKHORENKO, O., *Introduction to JavaServer Faces*, disponível em <http://onjava.com/pub/a/onjava/2004/04/07/jsf.html>, 2004, acessado em 13 de setembro de 2012.
- [34] LUCKOW, D. H., MELO, A. A., *Programação Java para a Web*. 1 ed. São Paulo, Novatec, 2010.
- [35] HIGHTOWER, R., *JSF for nonbelievers: The JSF application lifecycle - Walk through the 6 phases of JSF's request processing lifecycle*, disponível em <http://www.ibm.com/developerworks/library/j-jsf2/>, acessado em 13 de setembro de 2012.
- [36] KAPETANAKIS, M., Developer Economics 2011 – Why app stores are a one-way street, disponível em <http://www.visionmobile.com/blog/2011/06/developer-economics-2011-why-app-stores-are-a-one-way-street/>, acessado em 13 de setembro de 2012.
- [37] “iOS Developer Program”, <https://developer.apple.com/programs/ios/>, 2012, acessado em 13 de setembro de 2012.
- [38] “7 Things You Should Know About Geolocation”, <http://net.educause.edu/ir/library/pdf/ELI7040.pdf>, acessado em 13 de setembro de 2012.
- [39] DJUKNIC, G. M., RICHTON, R. E., *Geolocation and Assisted GPS*, disponível em <http://cens.ucla.edu/~mhr/cs219/location/djunkic01.pdf>, 2001, acessado em 13 de setembro de 2012.

- [40] “Processes and Threads”,
<http://developer.android.com/guide/components/processes-and-threads.html>,
2012, acessado em 13 de setembro de 2012.
- [41] “Doador Sangue Bom – Google Play”,
<https://play.google.com/store/apps/details?id=com.DoarSangue.Android>, 2012,
acessado em 13 de setembro de 2012.