UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FERNANDO HENRIQUE RODRIGUES

SISTEMA PARA IGREJAS EVANGÉLICAS MULTICONGREGADAS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2015

FERNANDO HENRIQUE RODRIGUES

SISTEMA PARA IGREJAS EVANGÉLICAS MULTICONGREGADAS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2015

ATA Nº: **275**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **FERNANDO HENRIQUE RODRIGUES**.

Às 13:30 hrs do dia 27 de novembro de 2015, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Rúbia E. O. Schultz Ascari (Convidada) e Lucilia Yoshie Araki (Convidada), para avaliar o Trabalho de Diplomação do aluno Fernando Henrique Rodrigues, matrícula 1147749, sob o título **Sistema para igrejas evangélicas multicongregadas**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 14:10 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Profa. Lucilia Yoshie Araki, M.Sc.
Convidada

Profa. Soelaine Rodrigues Ascari, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

RODRIGUES, Fernando Henrique. Sistema para igrejas evangélicas multicongregadas. 2015. 41f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2015.

Uma igreja, definida como um conjunto de pessoas que compartilham dos mesmos princípios religiosos, realiza atividades que incluem celebrações comunitárias, escolas doutrinais, venda de produtos religiosos entre outros. As diversas atividades realizadas por essas igrejas, muitas das quais envolvem receitas e despesas, que necessitam ser devidamente gerenciadas para que haja um adequado controle Considerando o volume de operações realizadas que envolvem recursos financeiros e da necessidade de uma correta e adequada prestação de contas, um sistema computacional pode auxiliar nestas atividades. A proposta deste trabalho se refere ao desenvolvimento de um sistema para igrejas evangélicas multicongregadas. Optou-se pelo desenvolvimento de um sistema para *web* em decorrência da facilidade de acesso e pelo uso da linguagem Java e tecnologias associadas para o desenvolvimento para *web* em função dos recursos oferecidos para implementação de interfaces consideradas ricas.

Palavras-chave: Java para web. Sistema para igrejas evangélicas. Aplicativo web.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE ABREVIATURAS

AJAX	*Asynchronous JavaScript and* XML
API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JPA	*Java Persistence API*
MVC	*Model, View, Controller*
ORM	*Object Relational Mapping*
RIA	*Rich Internet Applications*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa do trabalho. No final do capítulo é apresentada a organização do texto, por meio de uma breve descrição dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

O evangelicalismo é um movimento cristão, que surgiu no século XVII, após a Reforma Protestante (WIKIPEDIA, 2015), tornando-se uma vertente organizada do cristianismo.

Igreja evangélica é uma denominação religiosa para igreja do tipo protestante. Hoje, no Brasil, o termo evangélico tem sido usado para se referir a todos os que pertencem ao cristianismo em geral e que não são católicos romanos (DICIONARIO INFORMAL, 2015). São consideradas igrejas evangélicas (DICIONARIO INFORMAL, 2015):

- Protestantes históricos (Anglicana ou Episcopal, Luterana, Renovada ou Presbiteriana);
- Igrejas dos movimentos pentecostais e neopentecostais (Assembleia de Deus, Universal do Reino de Deus, Sara Nossa Terra, Igreja da Graça, Igreja do Evangelho Quadrangular, Igreja Batista Betel e outras);
- Igrejas emergentes e comunidades dos mais variados tipos.

Sendo uma entidade religiosa, uma igreja evangélica pode possuir muitos membros, e a igreja pode ser dividida em congregações. Além dos cultos periódicos é comum que a igreja preste serviços aos seus membros, como: escolas bíblicas, cursos teológicos e eventos (festas, batismos, almoços, encontros e congressos, entre outros).

Na igreja Assembleia de Deus de São Lourenço do Oeste, no Estado de Santa Catarina, o controle desses trabalhos é, atualmente, feito de forma manual, com preenchimento de livros caixas, fichas e recibos impressos. Essa forma de gestão ocorre em muitas igrejas desse tipo.

Esse campo eclesiástico de São Lourenço possui uma igreja sede e vinte congregações, nelas os departamentos trabalham da seguinte forma: a tesouraria efetua o recebimento de contribuições e ofertas, preenche um recibo manuscrito com os dados dos

contribuintes e os envia para arquivamento na igreja sede. Da mesma forma, o departamento da secretaria preenche uma ficha para cadastramento dos membros da igreja ou das congregações. O departamento de ensino possui um controle básico em editor de texto e documentos impressos com os dados de alunos e venda de material de estudo. A diretoria da igreja, juntamente com o Pastor Presidente do campo eclesiástico realiza duas reuniões anuais, que são as assembleias ordinárias, as quais são registradas em livro ata de forma manuscrita.

Diante deste contexto, percebeu-se que um sistema computacional que permita um controle automatizado e mais transparente contribuirá para facilitar a realização dessas tarefas. Um sistema *web* facilita o acesso para registro das atividades em todos os campos de trabalho da igreja, pois a igreja está em expansão e possui as congregações geograficamente afastadas.

1.2 OBJETIVOS

O objetivo geral está relacionado ao resultado principal que é esperado da realização deste trabalho. E os objetivos específicos complementam o objetivo geral em termos de funcionalidades do sistema.

1.2.1 Objetivo Geral

Implementar um sistema *web* para igrejas evangélicas visando auxiliar no gerenciamento e controle de congregações e dos seus respectivos departamentos.

1.2.2 Objetivos Específicos

- Prover uma forma de controle das atividades realizadas nas secretarias das igrejas evangélicas visando o registro dos membros da igreja, famílias e visitantes, da matriz, regionais, setores e congregações, cargos e funções.

- Definir mecanismos para gerenciamento da agenda das atividades da igreja, da escola bíblica e da faculdade teológica possibilitando o registro de cultos, cursos, conteúdo de cursos e controle de presença nos cursos.
- Possibilitar o registro de patrimônio da igreja, facilitando a identificação e a localização de cada item cadastrado.
- Prover um controle financeiro, com os respectivos lançamentos de contas de crédito e débito dos lançamentos realizados.
- Fornecer relatórios de acompanhamento, histórico e listagens dos diversos controles e registros realizados pelo sistema.

1.2 JUSTIFICATIVA

A falta de praticidade percebida pelos coordenadores da igreja Assembleia de Deus de São Lourenço do Oeste, para a qual o sistema está sendo proposto, no uso de fichas impressas, recibos e de livros caixa preenchidos de forma manuscrita é a justificativa principal para a realização deste trabalho. Para as pessoas que realizam os registros financeiros e os cadastros, a utilização de material impresso nem sempre é eficiente. Para o Pastor Presidente do campo eclesiástico e a diretoria geral, responsáveis pela coordenação da igreja e suas congregações, a manipulação do material impresso arquivado para elaboração de relatórios de resultados e para a conferência e acompanhamento dos lançamentos financeiros da igreja da maneira como tem sido realizada é uma tarefa lenta e passível de erros.

Para a coordenação da igreja e para as pessoas envolvidas com o operacional da mesma, um sistema *web* tornará mais fácil os lançamentos da secretaria, da tesouraria e do departamento de ensino. Bem como, o acompanhamento dessas atividades e dos resultados pela diretoria. Além disso, não será mais necessária a existência de um arquivo físico para armazenar os materiais impressos utilizados atualmente, sendo muito mais prática a busca pelas informações no banco de dados do sistema.

A justificativa de aplicabilidade do resultado deste trabalho se fundamenta na necessidade percebida de facilitar a gestão da igreja pela coordenação e demais pessoas envolvidas no processo. Em termos de tecnologias, a escolha de implementação de um sistema para *web* decorre da facilidade de acesso possibilitada por meio da Internet.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. Este é o primeiro e apresenta as considerações iniciais, o objetivo e a justificativa do trabalho. O Capítulo 2 apresenta o referencial teórico. No Capítulo 3 estão os materiais e o método. Os resultados da realização deste trabalho são apresentados no Capítulo 4. Por fim está a conclusão, seguida das referências bibliográficas.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico que fundamenta as tecnologias utilizadas no desenvolvimento do aplicativo que é para ambiente Internet.

2.1 APLICAÇÕES WEB

BENJAMIN *et al*. (2010) destacam que entre as evoluções técnicas ocorridas desde 1995 (últimos 15 anos a partir de 2010) uma é notável: o uso da *web* para apresentar interfaces de aplicações ao usuário. Ao transformar tais aplicações *web*, o software pode ser construído usando um padrão amplamente disponível para apresentar a interface com o usuário dessas aplicações que é *HyperText Markup Language* (HTML) apresentado em um navegador *web* e usando um protocolo padrão que é o *Hypertext Transfer Protocol* (HTTP) utilizado para troca de mensagens entre cliente e servidor.

O uso de HTML e HTTP apresenta uma série de benefícios (BENJAMIN *et al*., 2010):

a) torna mais fácil construir aplicações no lado cliente e que executará uma ampla variedade de sistemas, uma vez que praticamente qualquer sistema operacional fornece uma ou várias ferramentas de software que suportam os padrões necessários.

b) torna muito fácil distribuir e atualizar aplicações;

c) a configuração requerida ao usuário é mínima;

d) o uso sistemático do padrão de portas *Transmission Control Protocol* (TCP 80 e 443) facilita o tratamento pelos *firewalls*.

Apesar dos benefícios, esses autores citam algumas desvantagens das aplicações *web* e algumas dessas desvantagens têm implicações relacionadas à segurança (BENJAMIN *et al*., 2010):

a) o protocolo HTTP não foi projetado para servir aplicações e várias substituições para características necessárias tiveram que ser adicionadas (tais como *cookies* para manter sessões);

b) a abordagem de entregar qualquer coisa utilizando a mesma porta TCP, evitando assim *firewalls* baseados nessas portas. Esse é um contratempo a partir do ponto de vista de segurança, uma vez que desconsidera o objetivo de *firewalls* serem prioritários;

c) distribuição massiva de aplicações *web* cria uma rede de aplicações relacionadas que compartilham máquinas, dados e bases de dados de uma forma que a vulnerabilidade de

uma determinada aplicação frequentemente gera problemas em todas as aplicações co-hospedadas;

d) a facilidade com que aplicações *web* podem ser desenvolvidas permite com que aplicações *web* mal projetadas e desenvolvidas sejam distribuídas pela Internet.

Como uma evolução das aplicações *web* tradicionais estão as denominadas ricas. As *Rich Internet Applications* (RIA) definem uma nova geração de aplicações *web* pela introdução de processamento no lado cliente e comunicação assíncrona entre cliente e servidor, possibilitando alta usabilidade e apresentando conteúdo mais atrativo ao usuário (DRIVER; VALDES; PHIFER, 2005, STEARN, 2007). Embora tecnologias assíncronas tais como *Asynchronous JavaScript* and XML (AJAX) tornam as RIAs responsivas, elas podem resultar em comportamento inesperado devido aos eventos não determinísticos realizados pelo usuário e respostas do servidor (MAEZAWA; WASHIZAKI; HONIDEN, 2012). Assim, esses autores ressaltam que é importante controlar como as RIAs se comportam de acordo com suas mudanças de estado interativas.

As RIAs provem uma experiência interativa e responsiva (LAWTON, 2008) para o usuário, com interface gráfica com o usuário rica e respostas rápidas (CAMERON, 2004). Essas características não estão disponíveis em páginas de aplicações *web* clássicas, consideradas assim as baseadas unicamente em HTML, com interatividade limitada e respostas lentas (CAMERON, 2004). RIAs remontam aplicações *desktop* que são entregues com funcionalidades baseadas em *web* (DISSANAYAKE; DIAS, 2014). Com a introdução de HTML 5 e *Cascading Style Sheets* (CSS 3), as RIAs tem se tornando mais proeminentes em termos de portabilidade e disponibilidade (CORREIA, 2013).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para a realização deste trabalho. Os materiais estão relacionados às tecnologias e ferramentas utilizadas e o método apresenta a sequência das principais atividades realizadas.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias que foram utilizadas para modelar e implementar o sistema.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Apache Tomcat	7.0.55	http://tomcat.apache.org/	Servidor *web* para a aplicação.
MySQL	5.6.17	http://www.mysql.com/	Banco de dados.
Java	8u60	http://www.oracle.com/technetwork/pt/java/index.html	Linguagem de programação.
MySQL Workbench	6.3	https://www.mysql.com/products/workbench/	Administração do banco de dados MySQL.
Eclipse Mars	4.5.0	http://www.eclipse.org/	Ambiente de desenvolvimento.
Spring Framework	4.0.1	http://projects.spring.io/spring-framework/	Framework para aplicações Java Web.
SB Admin 2	1.0.7	http://startbootstrap.com/template-overviews/sb-admin-2/	Template Bootstrap gratuito, interface da aplicação.
jQuery	1.11.3	https://jquery.com/	Biblioteca JavaScript
Hibernate	5.0.2	http://www.hibernate.org/	Efetuar o mapeamento objeto-relacional e a persistência dos dados.
JSON	20150729	http://www.json.org/	Subconjunto da notação de objeto de JavaScript, usado para escrever objetos em JavaScript.

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

A seguir são apresentadas as atividades realizadas para a modelagem do sistema para igrejas evangélicas multicongregadas. As atividades realizadas foram de levantamento e

modelagem de requisitos e de implementação das funcionalidades básicas de um cadastro. A implementação teve como objetivo definir os recursos das tecnologias escolhidas para uso e apresentar a forma de implementação das operações de inclusão, exclusão, consulta e alteração realizadas nos cadastros.

a) Levantamento de requisitos

O levantamento dos requisitos iniciou com o pastor responsável por três congregações da igreja Assembleia de Deus da cidade de São Lourenço do Oeste, no Estado de Santa Catarina, fornecendo a visão geral das atividades operacionais da igreja, com os principais conceitos envolvidos e as necessidades de gestão e controle. Essa visão foi registrada como uma estrutura de diretórios contendo as principais funcionalidades (definidas como agrupamento macro de funcionalidades). A esses agrupamentos foram associadas as funcionalidades pretendidas para o sistema organizados em cadastro, lançamentos e relatórios, no que coubesse a cada agrupamento macro de funcionalidades. Dessa visão foram extraídos os requisitos principais e outros foram identificados posteriormente. Os requisitos foram organizados em funcionais e não funcionais.

b) Análise e projeto do sistema

Com base nos requisitos foram definidos os casos de uso do sistema. Os casos de uso foram documentados gerando informações para a definição do banco de dados e das classes. Digramas de classes e de entidades e relacionamentos do banco de dados, com a elaboração das tabelas e dos seus campos, tipo e tamanho de dados também foram definidos.

c) Implementação

A implementação foi realizada utilizando a ferramenta Eclipse Mars.

d) Testes

Os testes foram informais e realizados à medida que a implementação ocorria, sendo aplicado testes unitários. Testes funcionais também foram aplicados para validação do sistema.

4 RESULTADO

Este capítulo apresenta o resultado deste trabalho que é a modelagem e implementação de um sistema especifico para gerenciamento de igrejas do tipo evangélica com múltiplas congregações. No capítulo também constam alguns códigos principais utilizados na implementação do sistema.

4.1 ESCOPO DO SISTEMA

O sistema modelado como resultado deste trabalho se destina ao auxílio nas atividades de gestão de igrejas do tipo evangélicas, as quais podem ter uma ou mais congregações. Como exemplos dessas igrejas estão as Igrejas Assembleias de Deus, Igreja do Evangelho Quadrangular e Igreja Batista Betel. A solução proposta considera o contexto apresentado a seguir.

A gestão de uma igreja evangélica é dividida por departamentos. Cada departamento possui funções distintas, sendo a secretaria responsável pelo registro e organização dos membros, cargos e funções e pela agenda de cultos e trabalhos. A tesouraria é responsável pelo controle de patrimônio, financeiro e o fluxo do caixa. E o departamento de ensino é o responsável pela escola bíblica e pelos cursos teológicos.

Se a igreja possui mais de uma congregação, o controle dos departamentos ocorre individualmente em cada uma das congregações que compõem a igreja. Além da gestão separada por departamentos, o campo eclesiástico (igreja matriz e congregações) é regido pelo Pastor Presidente e por uma diretoria geral. Quando existem múltiplas congregações, há um Pastor responsável por elas e pelos seus departamentos.

4.2 MODELAGEM DO SISTEMA

A Figura 1 apresenta uma visão geral do sistema agrupado em quatro grandes segmentos. Secretaria e Tesouraria estão relacionados as atividades burocráticas e financeiras. Em Ensino estão as atividades da escola bíblica e da faculdade de teologia (esse é um curso superior aberto ao público na modalidade de ensino à distância). Departamentos estão

relacionados aos agrupamentos de participantes como crianças, jovens, adultos e idosos e outras funções realizadas.

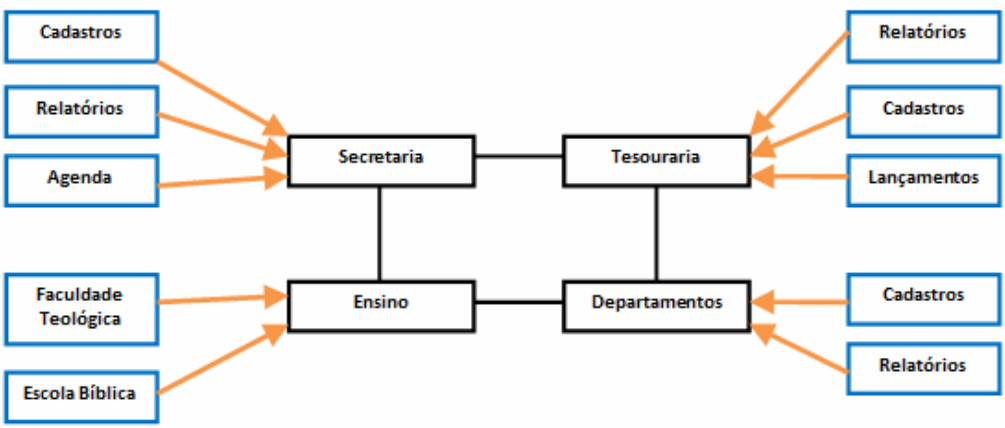

Figura 1 – Agrupamentos de funcionalidades do sistema

O Quadro 2 apresenta os agrupamentos de funcionalidades (acesso, secretaria, agenda, tesouraria, patrimônio, faculdade teológica, escola bíblica e departamentos) e para cada agrupamento são definidos os requisitos principais pretendidos para o sistema.

VISÃO GERAL DO SISTEMA ECCLESIA

ACESSO
- 1 - Cadastro de usuários
  - 1.1 - Permissões de Acesso

SECRETARIA

1 - Cadastros
- 1.1 - Cadastro da Sede/Matriz
- 1.2 - Cadastro das Regionais/Setores
- 1.3 - Cadastro das Congregações
- 1.4 - Cadastro de Cargos e Funções Eclesiásticas
- 1.5 - Cadastro de Situações
- 1.6 - Cadastro de Membros
  - 1.6.1 - Membro - Cargo - Situação - Congregação
- 1.7 - Cadastro de Famílias
- 1.8 - Cadastro de Visitantes

2 - Relatórios
- 2.1 - Relatórios das Congregações
- 2.2 - Relatórios de Membros
- 2.3 - Relatório de Famílias
- 2.4 - Relatórios de Visitantes

3 - Agenda
- 3.1 - Registro dos trabalhos na Agenda (por congregação)

TESOURARIA
1 - Cadastros
- 1.1 - Cadastro do Plano de Contas
- 1.2 - Cadastro de Produtos/Serviços
- 1.3 - Cadastro de Bancos e Caixas
- 1.4 - Cadastro de Fornecedores
- 1.5 - Cadastro de Clientes
- 1.6 - Cadastro de Patrimônios

2 - Lançamentos
- 2.1 - Transferências
  - 2.1.1 - Entre Contas
  - 2.2.2 - Lançamento Simples (Crédito ou Débito)
- 2.2 - Lançamento de Receitas
- 2.3 - Contas a Receber/Recebidas
- 2.4 - Lançamento de Despesas
- 2.5 - Contas a Pagar/Pagas

3 - Relatórios
- 3.1 - Relatório de Contas a Receber/Recebidas
- 3.2 - Relatório de Contas a Pagar/Pagas
- 3.3 - Relatório Demonstrativo de Resultados (Plano de Contas)
- 3.4 - Extrato de Bancos e Caixas
- 3.5 - Relatório de Patrimônios

ENSINO

Faculdade teológica
1 - Cadastros
- 1.1 - Cadastro de Níveis
- 1.2 - Montagem da Classe (Nível, Professor, Aluno)
- 1.3 - Criação da Grade Curricular
- 1.4 - Lançamento de Presença

2 - Relatórios
- 2.1 - Alunos
- 2.2 - Classes
- 2.3 - Presenças

Escola bíblica
1 - Cadastros
- 1.1 - Cadastro de Escola
- 1.2 - Montagem de Classe (Escola; Professor; Aluno)
- 1.3 - Criação da Grade Curricular
- 1.4 - Lançamento de Presença

2 - Relatórios
- 2.1 - Alunos
- 2.2 - Classes
- 2.3 - Presenças

DEPARTAMENTOS
1 - Cadastros
- 1.1 - Tipos de Departamentos
- 1.2 - Montagem do Grupo (Diretoria - Participantes)

2 - Relatórios
- 2.1 - Departamentos

Quadro 2 – Visão geral do sistema

O diagrama de entidades e relacionamentos é apresentado em partes para facilitar a visualização. A Figura 2 apresenta o diagrama de entidades e relacionamentos que representa o banco de dados da aplicação da funcionalidade de acesso ao sistema e controle de permissões.

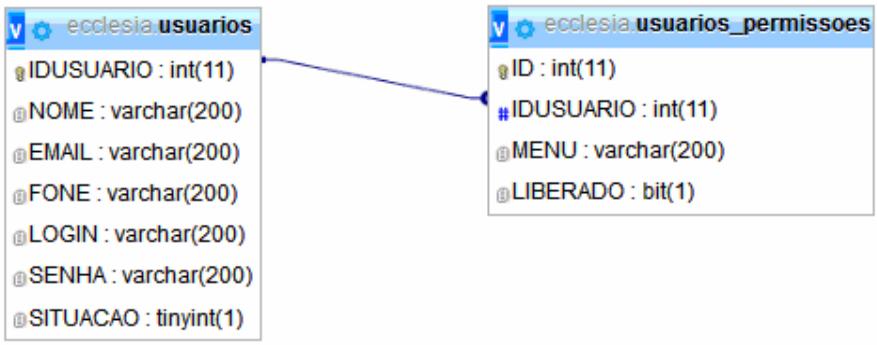

Figura 2 – Diagrama de entidades e relacionamentos do banco de dados: acesso e permissões

Na Figura 3 estão as entidades de banco de dados das funcionalidades relacionadas à secretaria.

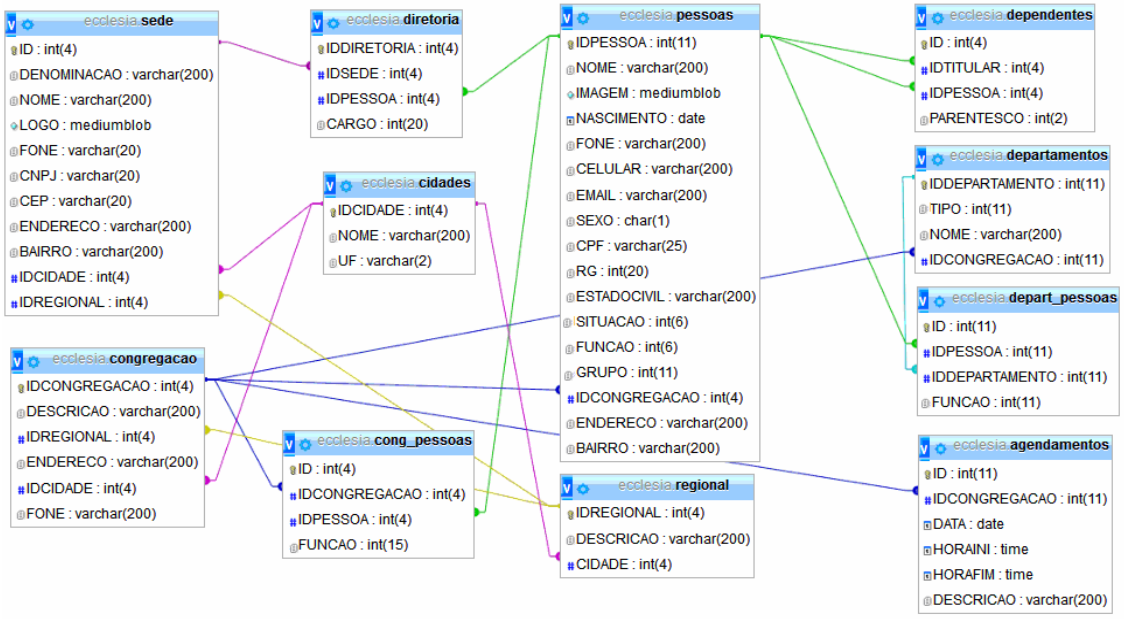

Figura 3 – Diagrama de entidades e relacionamentos do banco de dados: secretaria

As entidades de banco de dados das funcionalidades relacionadas ao financeiro são apresentadas na Figura 4.

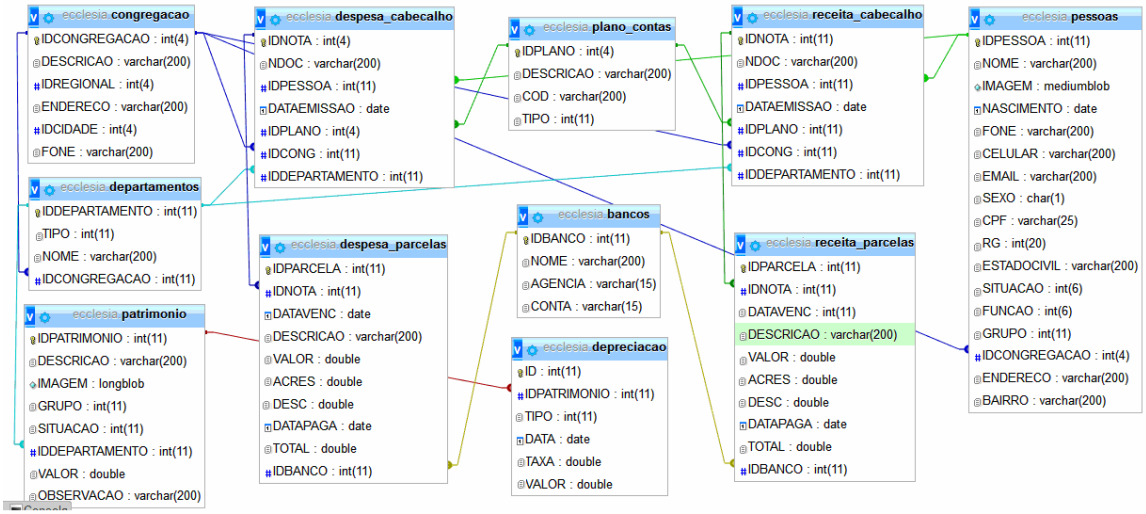

Figura 4 – Diagrama de entidades e relacionamentos do banco de dados: financeiro

A Figura 5 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação das funcionalidades relacionadas ao departamento de ensino (escola bíblica e faculdade teológica).

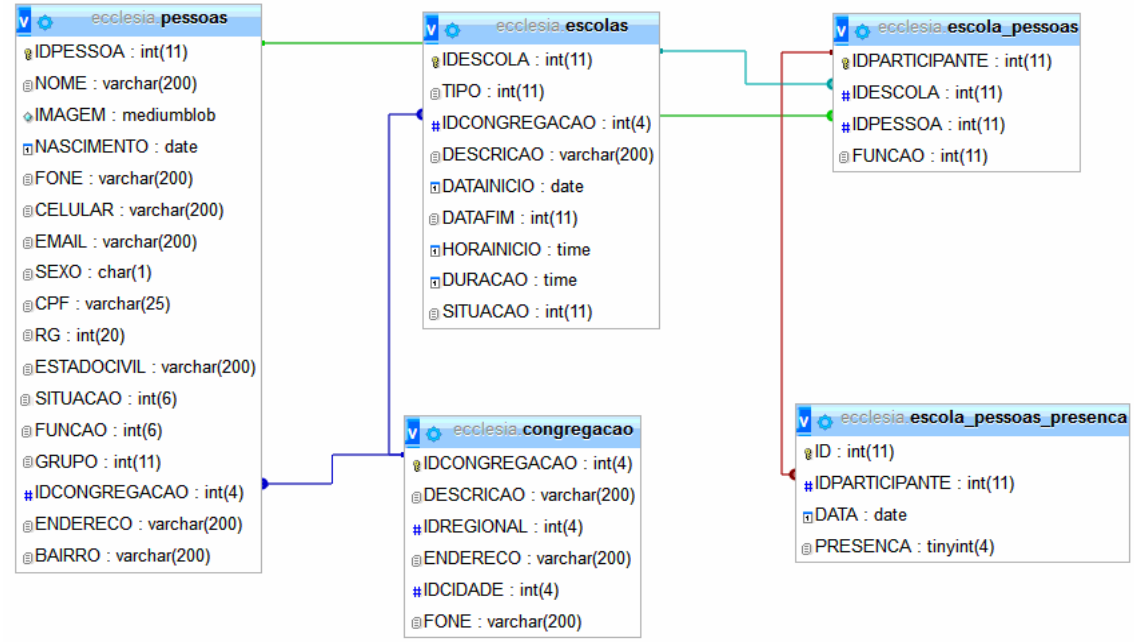

Figura 5 – Diagrama de entidades e relacionamentos do banco de dados: ensino

A Figura 6 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação das funcionalidades relacionadas aos departamentos gerais, como: infantil, jovem, missão e círculo de oração.

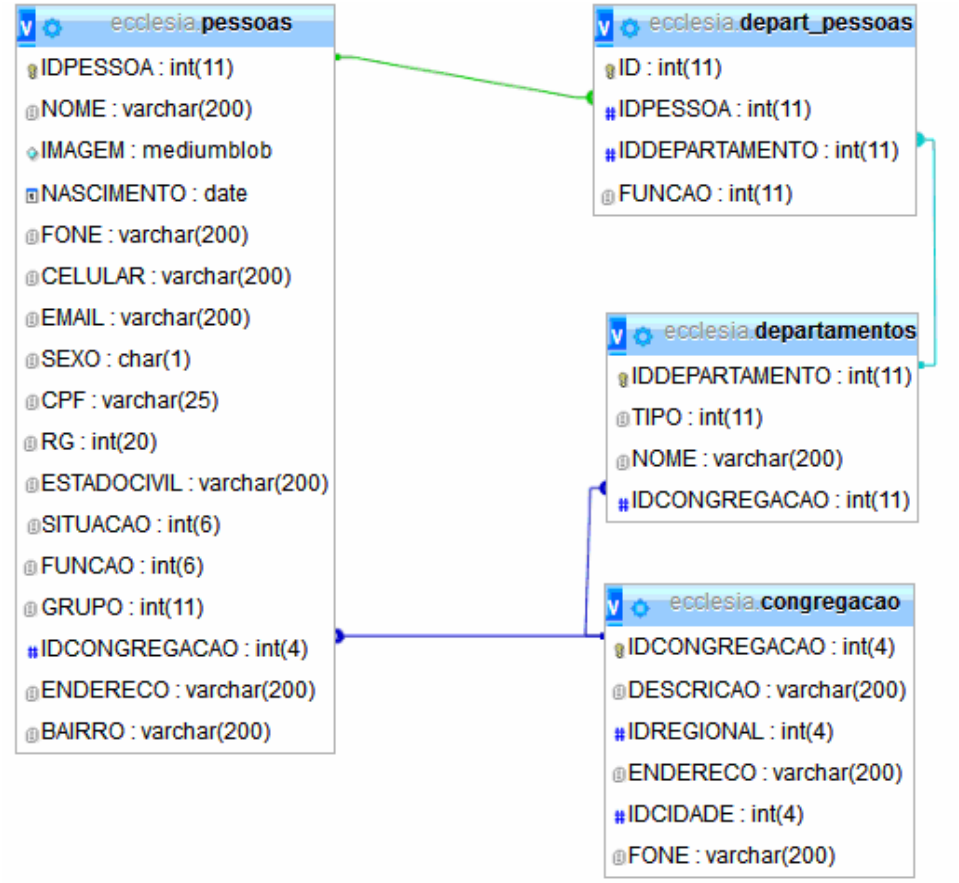

Figura 6 – Diagrama de entidades e relacionamentos do banco de dados: departamentos

4.3 APRESENTAÇÃO DO SISTEMA

O leiaute do sistema é composto por três setores: o setor superior contém o nome da igreja administradora e o *login* do sistema; o setor lateral esquerdo contém o menu com os módulos e suas funções; e o setor central apresenta o conteúdo da página que está sendo navegada. Os dois primeiros setores são estáticos e o segundo é alterado de acordo com a opção selecionada no menu apresentado no setor lateral esquerdo.

A Figura 7 apresenta os três setores nos quais a página foi organizada. No setor central está a área de “edição” dos formulários.

Figura 7 – Página inicial

O setor lateral esquerdo (módulos do sistema) é expansível e mostra somente as funções do módulo que está atualmente selecionado. Ficando, assim, mais prático para o usuário trabalhar, evitando sobrecarga decorrente de vários itens do menu que podem ser apresentados, como mostra a Figura 8.

Figura 8 – Menus do sistema

A Figura 9 mostra apenas o setor central, com o conteúdo da página de congregações. Nessa área o usuário visualiza as congregações já cadastradas e também pode realizar as operações básicas de cadastro como inclusão, alteração e exclusão.

Figura 9 – Lista de congregações

Ao clicar no botão “Nova Congregação”, o sistema abrirá o formulário de cadastro com os campos necessários para a inclusão de uma nova congregação, juntamente com os botões para confirmar ou cancelar, como mostra a Figura 10.

Figura 10 – Cadastro de congregações

Se o usuário clicar no botão “Confirmar” sem informar os campos obrigatórios, o sistema fará validação e apresentará uma mensagem para o usuário informar o campo obrigatório antes de salvar, como mostra a Figura 11.

Figura 11 – Campos obrigatórios

Se os campos forem válidos, o sistema armazenará o novo registro e mostrará a lista de congregações cadastradas atualizada, como mostrado na Figura 12.

Figura 12 – Lista atualizada

A operação de alteração é semelhante à de inclusão, sendo que, para alterar um registro desejado o usuário deve clicar no *link* "Alterar" ao final da linha da lista de congregações. Dessa forma, o sistema fará o mesmo procedimento utilizado na inclusão, mas, já trará o formulário preenchido com os dados da congregação selecionada. O processo de validação também é o mesmo. Para salvar as alterações, os campos obrigatórios devem estar preenchidos. Para realizar a operação de exclusão, o usuário deve clicar no link "Excluir" da linha referente ao registro desejado.

As telas do controle financeiro (receitas e despesas) são divididas em três áreas, conforme é mostrado na Figura 13. Nessa tela, na primeira parte (área 01) o usuário deverá informar os dados do cabeçalho da nota, tais como: data de emissão; cliente (que pode ser pessoa física ou jurídica) e congregação a qual pertence à referida receita.

Após preencher o cabeçalho é necessário adicionar as parcelas dessa nota, informando a descrição, o valor, o acréscimo e desconto, se houver, conforme mostra a Figura 13 na região destacada como área 02.

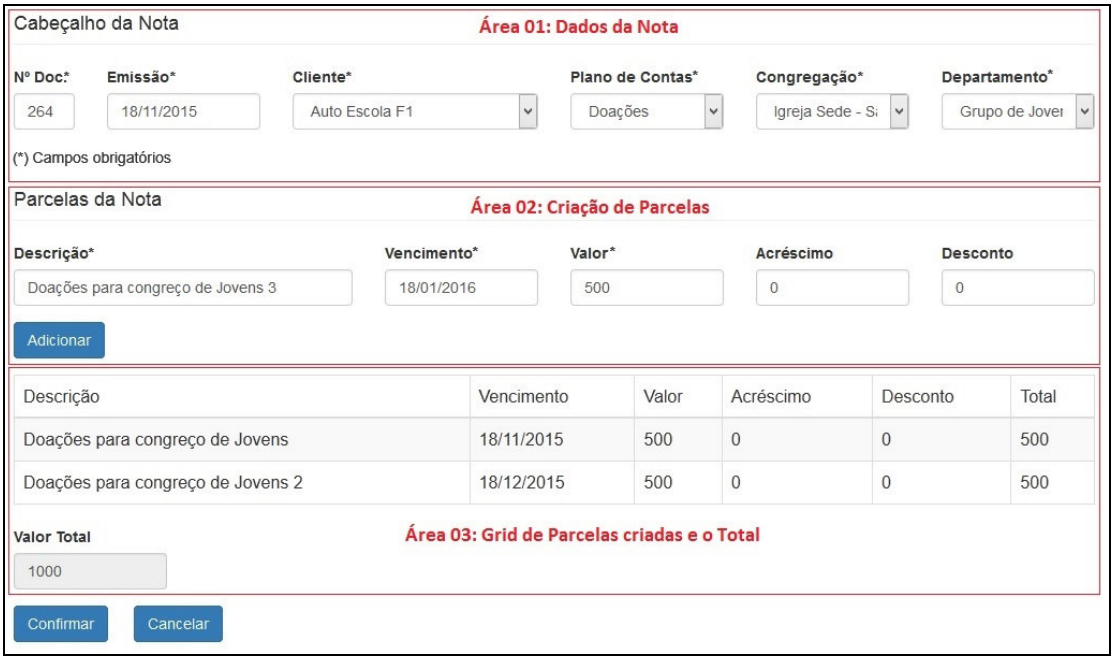

Figura 13 – Lançamento de receitas

Após preencher (área 02) e clicar no botão "Adicionar" o sistema cria um registro no *grid* de parcelas (Área 03) com os dados que foram informados, atualiza o valor total da nota e limpa os campos da área 02 para que o usuário possa adicionar outra parcela sem a necessidade de apagar manualmente os dados da anteriormente lançada, como mostra a Figura 14.

Figura 14 – Lançamento de Receita atualizado

Depois de confirmar o lançamento, o sistema direciona a visualização para a tela de "Contas a Receber/Recebidas", com uma lista de parcelas lançadas, para que o usuário possa usar as funções: receber, alterar, excluir e adicionar uma nova receita. Conforme mostra a Figura 15.

Figura 15 – Contas a receber/recebidas

Ao clicar no *link* "Receber", o sistema apresenta a tela de recebimento com os dados da parcela selecionada. Nesse momento é possível adicionar desconto ou acréscimo para atualizar o valor total da parcela. Nesta tela o usuário deverá escolher a conta (caixa ou bancária) para o recebimento e a data do mesmo, conforme mostra a Figura 16.

Figura 16 – Recebimento da parcela

Após confirmar o recebimento, o sistema mostra a lista de parcelas na tela de contas a receber/recebidas atualizada com o campo data de recebimento preenchido, conforme mostra a Figura 17.

Contas a Receber/Recebidas

Nova Receita | Voltar ao Início

Doc	Cliente	Descrição	Emissao	Vencimento	Valor	Acréscimo	Desconto	Total	Recebimento			
264	Auto Escola F1	Doações para congreço de Jovens	2015-11-18	2015-11-18	500.0	0.0	0.0	500.0	2015-11-18	Receber	Alterar	Excluir
264	Auto Escola F1	Doações para congreço de Jovens 2	2015-11-18	2015-12-18	500.0	0.0	0.0	500.0		Receber	Alterar	Excluir
264	Auto Escola F1	Doações para congreço de Jovens 3	2015-11-18	2016-01-18	500.0	0.0	0.0	500.0		Receber	Alterar	Excluir

Figura 17 – Parcelas a receber/recebidas atualizada com data de recebimento

Esse processo também serve para incrementar o saldo das contas, efetuando o controle do fluxo de caixa, conforme mostra a Figura 18, no relatório do extrato de recebimentos.

Extrato de Recebimentos

Conta - Caixa Sede

Histórico do Lançamento	Data Rec.	Total
Doações para congreço de Jovens	18/11/2015	500.0
Doações para congreço de Jovens 2	18/11/2015	500.0

Saldo: 1.000,00

Figura 18 – Extrato de recebimentos Caixa Sede

Para o controle de contas a pagar o sistema trabalha no mesmo formato que o apresentado de contas a receber, apenas mudando o contexto para despesas.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Como apresentado na Seção 4.3, o leiaute do sistema é composto por três setores, isso é feito por meio do uso de um tema do Bootstrap, o SB Admin 2. Esse é um tema para administrador para sistemas *web*.

Para facilitar o desenvolvimento, foi utilizada uma página modelo, chamada "template.tag". A Listagem 1 apresenta um pequeno trecho desse código. Essa página contém o leiaute básico do sistema que carrega os códigos fonte dos componentes do setor superior e lateral esquerdo.

```
<li><a href="#"><i class="fa fa-folder-open-o"></i> Secretaria<span
class="fa arrow"></span></a>
<ul class="nav nav-second-level">

<li><a href="#"><i class="fa fa-book"></i> Cadastros <span class="fa
arrow"></span></a>
<ul class="nav nav-third-level">
   <li><a href="/secretaria/cadastro/cidade/">Cidades</a></li>
   <li><a href="/secretaria/cadastro/regional/">Regionais</a></li>
   <li><a href="/secretaria/cadastro/congregacao/">Congregações</a></li>
   <li><a href="/secretaria/cadastro/pessoa/">Membros</a></li>
   <li><a href="#">Famílias</a></li>
   <li><a href="#">Visitantes</a></li>
</ul> <!-- /.nav-third-level --></li>
<li><a href="#"class="fa fa-file-o"> Relatórios <span ></span></a>
<ul class="nav nav-third-level">
  <li><a href="#">Congregações</a></li>
  <li><a href="#">Membros</a></li>
  <li><a href="#">Famílias</a></li>
  <li><a href="#">Visitantes</a></li>
</ul> <!-- /.nav-third-level --></li>
<li><a href="#">Agenda <span class="fa arrow"></span></a>

<ul class="nav nav-third-level">
<li><a href="#">Registro por Congregações</a></li>

                                                    </ul> <!-- /.nav-
third-level --></li>
                                            </ul> <!-- /.nav-second-level -->
</li>
```

Listagem 1 – Trecho do cógido da template.tag

Utilizando essa *template* é necessário fazer somente a criação de páginas para o conteúdo do setor central das funcionalidades do sistema. Para cada rotina, o sistema possui duas páginas (index.jsp e o cadastro.jsp), conforme está destacado na Figura 19.

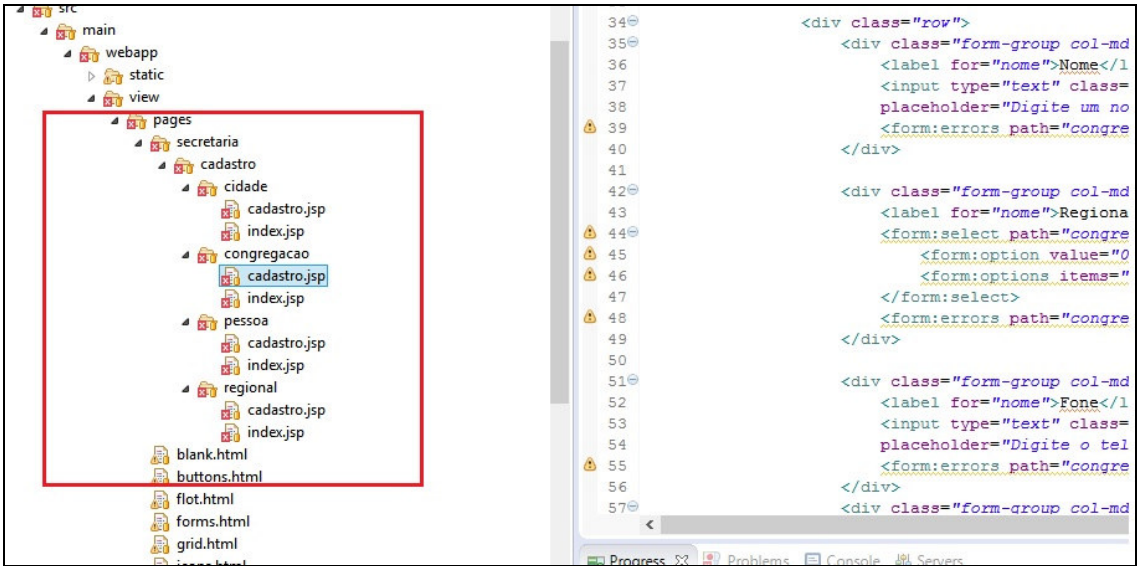

Figura 19 – Páginas do setor central

O sistema foi desenvolvido usando o Spring Framework, com o padrão de arquitetura de software *Model-View-Controller* (MVC). Assim, há as páginas de interação com o usuário (*views*) que são: O index.jsp e o cadastro.jsp. O index.jsp (Listagem 2) da rotina de cadastro de regionais, serve para listar os dados cadastrados, solicitar inclusão, alteração e exclusão dos registros por meio de requisições enviadas para a classe RegionalController da aplicação.

```
<layout:template>
      <jsp:body>

<div class="row">
          <div class="col-lg-12">
            <h2 class="page-header">Regionais</h2>
          </div>

</div>
      <div class="row">
         <div class="col-lg-12">
           <a href="novo/" class="btn btn-primary">Nova Regional</a>
              <span style="padding-left:20px"></span>
              <a href="/" class="btn btn-primary">Voltar ao Íncio</a>
                        </br></br>
          </div>
</div>

<table
 class="table table-striped table-bordered table-hover table-responsive">
        <thead>
           <tr>
             <td>Código</td>
             <td>Nome</td>
             <td>Cidade Sede</td>
             <td>UF</td>
             <td></td>
             <td></td>
           </tr>
         </thead>
```

```
        <tbody>
            <c:forEach items="${regionais}" var="regional">
                <tr>
                  <td>${regional.codigo}</td>
                  <td>${regional.nome}</td>
                  <td>${regional.cidade.nome}</td>
                  <td>${regional.cidade.uf}</td>
<td><a
href="/secretaria/cadastro/regional/${regional.codigo}">Alterar</a></td>
<td><a
href="/secretaria/cadastro/regional/${regional.codigo}/excluir">Excluir</a
></td>
                </tr>
            c:forEach>

         </tbody>
</table>

</jsp:body>
</layout:template>
```

Listagem 2 – index.jsp módulo secretaria – regionais

A página cadastro.jsp possui o formulário que faz o envio dos dados para inclusão e alteração utilizando de uma requisição *post* para a classe RegionalController, conforme mostrado na Listagem 3.

```
<layout:template>
      <jsp:body>
<div class="row">
                              <div class="col-lg-12">
                                    <h2 class="page-header">Regionais</h2>
                              </div>

                        </div>

                                    <div class="row">
                              <div class="col-md-12">
                                    <spring:hasBindErrors name="regional">
                                    <ul>
                                          <c:forEach var="error"
items="${errors.allErrors}">
                                          <li><spring:message
code="${error.code}"

      text="${error.defaultMessage}" /></li>
                                          </c:forEach>


                                    </ul>
                                    </spring:hasBindErrors>
                              </div>
                        </div>

                  <form action="/secretaria/cadastro/regional/"
method="post">

                        <div class="row">
                              <div class="form-group col-md-5">
                                    <label for="nome">Nome</label>
```

```
                                        <input type="text" class="form-control"
name="nome" id="nome"
                                        placeholder="Digite um nome..."
value="${regional.nome}">
                                        <form:errors path="regional.nome" />
                                    </div>

                                    <div class="form-group col-md-3">
                                        <label for="nome">Cidade Sede</label>
                                        <form:select
path="regional.cidade.codigo" class="form-control">
                                        <form:option value="0" label="Selecione
uma Cidade" />
                                        <form:options items="${cidades}"
itemValue="codigo" itemLabel="nome" />
                                    </form:select>
                                    <form:errors path="regional.cidade"/>
                                    </div>
                                </div>

                                    <input type="hidden"
value="${regional.codigo}" name="codigo"
                                id="codigo">
                                    <button type="submit" class="btn btn-
primary">Confirmar</button>
                                    <span style="padding-left: 20px"></span>
                                    <a href="/secretaria/cadastro/regional/"
class="btn btn-primary">Voltar</a>
                            </form>
</jsp:body>
</layout:template>
```

Listagem 3 – cadastro.jsp módulo secretaria – regionais

Por meio da anotação @RequestMapping na classe "RegionalController" (Listagem 4) a aplicação entende que as requisições que vem pela *Uniform Resource Locator* (URL) ("/secretaria/cadastro/regional") são para os métodos desta classe. Neste *controller* também é instanciada a classe Regionais (pacote *Models*), para adicionar os atributos desta tabela.

```
@Controller
@RequestMapping("/secretaria/cadastro/regional")
public class RegionalController {

	@Autowired
	private Cidades cidadeRepository;

	@Autowired
	private Regionais repository;

	@RequestMapping(value = "/", method = RequestMethod.GET)
	public String index(Model model) {
		populaView(model);
		return "pages/secretaria/cadastro/regional/index";
	}

	@RequestMapping(value = "novo/", method = RequestMethod.GET)
	public String inserir(Model model) {
		model.addAttribute("cidades", cidadeRepository.todas());
```

```
            model.addAttribute("regional", new Regional());
            return "pages/secretaria/cadastro/regional/cadastro";
      }

      @RequestMapping(value = "/", method = RequestMethod.POST)
      public String salvar(@Valid Regional regional, BindingResult erros,
RedirectAttributes redirect, Model model) {
            if (erros.hasErrors()) {
                  return "pages/secretaria/cadastro/regional/cadastro";
            }
            if (regional.getCodigo() != null) {
                  repository.alterar(regional);
            } else {
                  repository.inserir(regional);
            }
            redirect.addFlashAttribute("mensagem", "Regional salva com
sucesso");
            return "redirect:/secretaria/cadastro/regional/";
      }

      @RequestMapping(value = "/{codigo}", method = RequestMethod.GET)
      public String alterar(@PathVariable Long codigo, Model model) {
            model.addAttribute("cidades", cidadeRepository.todas());
            model.addAttribute("regional",
repository.findByCodigo(codigo));
            return "pages/secretaria/cadastro/regional/cadastro";
      }

      @RequestMapping(value = "/{codigo}/excluir", method =
RequestMethod.GET)
      public String excluir(@PathVariable Long codigo, Model model) {
            repository.excluir(codigo);

            return "redirect:/secretaria/cadastro/regional/";
      }

      private void populaView(Model model) {
            model.addAttribute("regionais", repository.todas());
            model.addAttribute("cidades", cidadeRepository.todas());
      }
}
```

Listagem 4 – RegionalController.java

Na Listagem 4 também é possível verificar que quando é requisitada a URL "/" (página index.jsp) será executado o método index(). Esse método chama outro método, o populaview() que faz a busca das regionais cadastradas pelo método todas(), que é um atributo do *repository* Regionais.

A classe Regionais.java (Listagem 5), do pacote de repositórios, é responsável por toda a lógica relacionada a manipular dados e recuperá-los do banco de dados. Classes anotadas com @Repository são automaticamente instanciadas, injetadas e gerenciadas dentro do Spring. Essa classe possui os métodos do EntityManager do *Java Persistence API* (JPA), que provê *Application Programming Interface* (API) para criar consultas, buscando, sincronizando e inserindo objetos no banco de dados.

O método todas() retorna uma lista (List<Regional>) de objetos do tipo Regional (classe do *model*), com todas as regionais cadastradas. Para isso ocorrer o atributo "em" (que é uma instância de EntityManager) cria e executa uma *query* que seleciona os dados do banco. O resultado é enviado para o método populaview() da classe RegionalController que é instanciado no método index() que retorna os dados para a *view* (index.jsp). Nessa página (Listagem 1) a taglib jstl "c:forEach" carrega os dados na tela, com os atributos selecionados.

```
@Repository
@Transactional
public class Regionais implements Serializable {

	private static final long serialVersionUID = 1L;

	@Autowired
	private EntityManager em;

	public List<Regional> todas() {
		Query query = em.createQuery("select r from Regional r");
		return (List<Regional>) query.getResultList();
	}

	public Regional findByCodigo(Long codigo) {
		return em.find(Regional.class, codigo);
	}

	public void inserir(Regional regional) {

		em.persist(regional);

	}

	public void alterar(Regional regional) {

		em.merge(regional);

	}

	public void excluir(Long codigo) {
		Regional regional = em.find(Regional.class, codigo);
		em.remove(regional);
	}
}
```

Listagem 5 – Repositorio: Regionais.java

Para gravar ou alterar os dados no banco, a classe RegionalController recebe as requisições da *view* cadastro.jsp e executa o método salvar() (Listagem 6). Esse método verifica se o código de registro no *model* Regional (tabela) está vazio ou preenchido, então envia os atributos do *model* Regional para um dos métodos do *repository*: alterar() ou inserir().

```
	@RequestMapping(value = "/", method = RequestMethod.POST)
	public String salvar(@Valid Regional regional, BindingResult erros,
RedirectAttributes redirect, Model model) {
```

```
            if (erros.hasErrors()) {
                  return "pages/secretaria/cadastro/regional/cadastro";
            }
            if (regional.getCodigo() != null) {
                  repository.alterar(regional);
            } else {
                  repository.inserir(regional);
            }
            redirect.addFlashAttribute("mensagem", "Regional salva com
sucesso");
            return "redirect:/secretaria/cadastro/regional/";
      }
```

Listagem 6 – método salvar()

No *repository* Regionais são utilizados os métodos do EntityManager (Listagem 7) para inserir (em.persist), alterar (em.merge) e excluir (em.remove) que executam as alterações no banco de dados, nos atributos da variável regional, que é uma classe do *model*.

```
      public void inserir(Regional regional) {

            em.persist(regional);

      }

      public void alterar(Regional regional) {

            em.merge(regional);

      }

      public void excluir(Long codigo) {
            Regional regional = em.find(Regional.class, codigo);
            em.remove(regional);
      }
```

Listagem 7 – métodos crud

No pacote *model.secretaria* está a classe Regional (Listagem 8), que representa uma tabela do banco de dados. A anotação @Entity informa ao Hibernate que esta classe é uma entidade e seus atributos devem ser mapeados para o banco de dados. O *repository* e o *controller* fazem uso desses atributos em seus métodos.

```
@Entity
@Table(name="regional")
public class Regional implements Serializable{
      private static final long serialVersionUID = 1L;

      @Id
      @GeneratedValue(strategy= GenerationType.IDENTITY)
      private Long codigo;
      private String nome;

      @ManyToOne
      @JoinColumn(name="codigo_cidade")
      private Cidade cidade;

      public Long getCodigo() {
```

```
		return codigo;
	}

	public void setCodigo(Long codigo) {
		this.codigo = codigo;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public Cidade getCidade() {
		return cidade;
	}

	public void setCidade(Cidade cidade) {
		this.cidade = cidade;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((codigo == null) ? 0 :
codigo.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Regional other = (Regional) obj;
		if (codigo == null) {
			if (other.codigo != null)
				return false;
		} else if (!codigo.equals(other.codigo))
			return false;
		return true;
	}
}
```

Listagem 8 – *Model*: Regional.java

O controle financeiro segue a mesma metodologia dos cadastros, tendo as páginas de edição, as *views*; os *controllers* para os métodos da aplicação e os *models* para as entidades do banco de dados. O que difere de um cadastro normal, além de uma complexidade maior nas operações, são os códigos das funções *javascript* para criação do grid de parcelas na tela de cadastro das receitas e despesas, conforme é mostrado abaixo na Listagem 9.

```
function adicionarParcela() {
            var valor = parseFloat($("#valor").val());
            var acrescimo = parseFloat($("#acrescimo").val());
            var desconto = parseFloat($("#desconto").val());

            if (!valor) {
                  valor = 0.0;
            }
            if (! acrescimo) {
                  acrescimo = 0.0;
            }
            if (!desconto) {
                  desconto = 0.0;
            }

            var parcela = {
                  codigo: codigo,
                  descricao: $("#descricao").val(),
                  vencimento: $("#vencimento").val(),
                  valor: valor,
                  acrescimo: acrescimo,
                  desconto: desconto,
                  total: valor + acrescimo - desconto
            }

            parcelas.push(parcela);
            codigo --;

            removeHidden();
            removeGrid();
            adicionarHidden();
            adicionarGrid();
            calculaValorTotal();
            limpaTela();
      }

      function limpaTela(){
            $("#descricao").val("");
            $("#vencimento").val("");
            $("#valor").val("");
            $("#acrescimo").val("");
            $("#desconto").val("");
      }

      function removeHidden() {
            $("input[name*='parcelas[']").remove();
      }

      function removeGrid() {
            $("#gridParcelas tbody").html("");
      }

      function adicionarHidden(){
            for (var i = 0; i < parcelas.length; i++) {
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].codigo", name: "parcelas[" + i +
"].codigo",}).val(parcelas[i].codigo).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].descricao", name: "parcelas[" + i +
"].descricao",}).val(parcelas[i].descricao).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
```

```
"].valor", name: "parcelas[" + i +
"].valor",}).val(parcelas[i].valor).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].acrescimo", name: "parcelas[" + i +
"].acrescimo",}).val(parcelas[i].acrescimo).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].desconto", name: "parcelas[" + i +
"].desconto",}).val(parcelas[i].desconto).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].vencimento", name: "parcelas[" + i +
"].vencimento",}).val(parcelas[i].vencimento).appendTo("form");
                  $("<input>").attr({type: "hidden", id: "parcelas[" + i +
"].total", name: "parcelas[" + i +
"].total",}).val(parcelas[i].total).appendTo("form");
            }
      }

      function adicionarGrid(){
            console.log(parcelas);
            for (var i = 0; i < parcelas.length; i++) {
                  $("#gridParcelas tbody").last().append($("<tr/>"));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].descricao));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].vencimento));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].valor));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].acrescimo));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].desconto));
                  $("#gridParcelas tbody
tr").last().append($("<td/>").text(parcelas[i].total));
            }
      }

      function calculaValorTotal(){
            var valor = 0.0;
            for (var i = 0; i < parcelas.length; i++) {
                  valor += parcelas[i].total;
            }
            $("#total").val(valor);
      }
```

Listagem 9 – Funções JavaScript para lançamento de parcelas de receitas

O método de alteração nos *controllers* das rotinas dos lançamentos (receitas ou despesas) também possui um diferencial, funções JSON para recuperação dos objetos do banco, ou seja, o *grid* de parcelas das notas lançadas, conforme é mostrado na Listagem 10.

```
@RequestMapping(value = "/{codigo}", method = RequestMethod.GET)
public String alterar(@PathVariable Long codigo, Model model) {
      Receita receita = repository.findByCodigo(codigo);

      JsonNodeFactory nodeFactory = new JsonNodeFactory(true);

      ObjectMapper mapper = new ObjectMapper();
      ArrayNode array = mapper.createArrayNode();

      SimpleDateFormat dt = new SimpleDateFormat("dd/MM/yyyy");
```

```
	for (ReceitaParcela parcela : receita.getParcelas()) {
		ObjectNode node = nodeFactory.objectNode();

		node.put("codigo", parcela.getCodigo());
		if (parcela.getVencimento() != null) {
			node.put("vencimento", dt.format(parcela.getVencimento()));
		}
		node.put("descricao", parcela.getDescricao());
		node.put("valor", parcela.getValor());
		node.put("acrescimo", parcela.getAcrescimo());
		node.put("desconto", parcela.getDesconto());
		node.put("total", parcela.getTotal());

		array.add(node);
	}

	model.addAttribute("parcelas", array.toString());

	model.addAttribute("receita", receita);
	return "pages/financeiro/lancamentos/receitas/cadastro";
}
```

Listagem 10 – Método alteração *controller* de receitas

5 CONCLUSÃO

O objetivo deste trabalho foi modelar um sistema para gerenciamento de igrejas evangélicas multicongregadas, abrangendo as rotinas executadas nesse tipo de entidade. Em complemento à modelagem, algumas telas de cadastros do módulo "Secretaria" foram desenvolvidas. Esses controles realizam as operações básicas (inclusão, atualização, exclusão e consulta) sobre as entidades: cidades, regionais, congregações e pessoas presentes na Figura 2 deste trabalho. A apresentação da implementação teve como objetivo fornecer exemplo da utilização das ferramentas e das tecnologias empregadas no desenvolvimento da codificação.

A aplicação foi desenvolvida para a plataforma *web*, pela facilidade de acesso e manutenção, utilizando recursos que a caracterizam como uma RIA. Para esse desenvolvimento foram utilizadas diversas ferramentas e tecnologias.

Uma destas ferramentas é o Hibernate, um *framework* para realizar o *Object Relational Mapping* (ORM) escrito na linguagem Java. O Hibernate visa reduzir a complexidade envolvida no desenvolvimento de aplicações que utilizam banco de dados relacional. Essa redução é atribuída à realização da intermediação entre o banco de dados e sua aplicação, desvinculando o desenvolvedor da necessidade de implementar instruções *Structured Query Language* (SQL) para recuperar ou persistir os dados do seu software. No modelo MVC, o trabalho é facilitado com o uso do Hibernate, pois basta utilizar uma anotação para informar que uma determinada classe do *Model* da aplicação é uma entidade e, a partir disso, o Hibernate cria uma tabela no banco de dados para esta classe.

Outra ferramenta utilizada é o Spring. Um *framework* de código aberto (*open source*), criado com o intuito de simplificar a programação em Java. O Spring atualmente possui diversos módulos como Spring Data (trata da persistência) e Spring Security (trata da segurança da aplicação).

O Spring MVC é conhecido como o principal *framework web* Java, possui uma aceitação boa pela comunidade e uma vasta documentação (que pode ser encontrada, inclusive, na página oficial https://spring.io/docs), facilitando o desenvolvimento e a solução de possíveis problemas. Uma das suas desvantagens está na complexidade inicial para uso, mas com uma busca no vasto material disponível gratuitamente na Internet é possível aprender o uso desse *framework*.

REEFERÊNCIAS

BENJAMIN, Kamara; BOCHMANN, Gregor v.; JOURDAN, Guy-Vincent; ONUT, Iosif-Viorel. **Some modeling challenges when testing rich internet applications for security.** Third International Conference on Software Testing, Verification, and Validation Workshops, IEEE Computer Society, 2010, p. 403-409.

CAMERON. O'Rourke. A look at rich internet application. **Oracle Magazine**. Jul./ago., 2004, p. 1-4.

CORREIA, Edward J. **What's next for HTML5?**. Intel Software Adrenaline, 2013. Disponível em: < https://software.intel.com/en-us/articles/whats-next-for-html5>. Acesso em: 18 ago. 2015.

DICIONÁRIO INFORMAL. **Evangélico**. Disponível em: <http://www.dicionarioinformal.com.br/evang%C3%A9lico/>. Acesso em: 14 ago. 2015.

DISSANAYAKE, Nalaka R.; DIAS, G.K.A. **Best practices for rapid application development of AJAX based Rich Internet Applications**. In: International Conference on Advances in ICT for Emerging Regions (ICTer), 2014, p. 63-66.

DRIVER, Mark; VALDES, Ray; PHIFER, Gene. Rich internet applications are the next evolution of the web, **Tech. report**, Gartner, 2005.

LAWTON, George. New ways to build rich Internet applications. **Computer**. Published by the IEEE Computer Society, p. 10-12, 2008.

MAEZAWA, Yuta; WASHIZAKI, Hironori; HONIDEN, Shinichi. **Extracting interaction-based stateful behavior in rich internet applications**. In: 16th European Conference on Software Maintenance and Reengineering, 2012, p. 423-428.

STEARN, Brent. XULRunner: a new approach for developing rich internet applications. **IEEE Internet Computing**, v. 11, p. 67-73, 2007.

WIKIPEDIA. **Evangelicalismo**. Disponível em: <https://pt.wikipedia.org/wiki/Evangelicalismo>. Acesso em: 14 ago. 2015.