UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

DENIS GIBIKOSKI

SISTEMA *WEB* PARA RESERVA DE SALÃO DE FESTAS EM CONDOMÍNIOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2019

DENIS GIBIKOSKI

SISTEMA *WEB* PARA RESERVA DE SALÃO DE FESTAS EM CONDOMÍNIOS

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Prof.ª Andreia Scariot Beulke

PATO BRANCO
2019

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

SISTEMA *WEB* PARA RESERVA DE SALÃO DE FESTAS EM CONDOMÍNIOS

por

DENIS GIBIKOSKI

Este trabalho de conclusão de curso foi apresentado em 09 de dezembro de 2019, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Andreia Scariot Beulke (orientadora), Mariza Miola Dosciatti e Vinicius Pegorini, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Andreia Scariot Beulke
Profa. Orientadora (UTFPR)

Mariza Miola Dosciatti
(UTFPR)

Vinicius Pegorini
(UTFPR)

Vinicius Pegorini
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Este trabalho teve como objetivo desenvolver um sistema *web* para reserva de salão de festas em condomínios. Um condomínio é destinado ao uso de uma área habitacional ou comercial e possui espaço de uso individual e espaço de uso comum pertencente aos diversos proprietários. O salão de festas é um exemplo de espaço de uso comum, que os condôminos utilizam para realizar eventos como, festas, reuniões, almoço ou jantar, por exemplo, entre familiares e amigos. Como esse espaço é de uso comum entre os condôminos, é importante haver um controle de reservas. Normalmente, as reservas são controladas pelo síndico, que é a pessoa responsável pelo condomínio. O condômino solicita uma reserva e o síndico a valida, verificando se o dia e o horário estão disponíveis. Como resultado desse trabalho foi desenvolvido um sistema *web* que possibilita que condôminos possam reservar o salão de festas e o síndico tenha um controle mais efetivo das reservas de um condomínio. Entre as tecnologias utilizadas no desenvolvido do trabalho estão a linguagem Java e o JavaServer Faces.

Palavras-chave: Condomínio. Salão de festas. JavaServer Faces.

ABSTRACT

This work aimed to develop a *web* system for reservation of ballroom in condominium. A condominium is a space intended for the use of a residential or commercial area and has a space for individual and common use that belongs to the several owners. The ballroom is an example of this kind of space, which the owners use to do events such as parties, meetings, lunch or dinner, for example, between their family and friends. These spaces are commonly used by the owners, thus it is important to have a reservation control. Usually the reservations are controlled by the liquidator, who is the person responsible for the condominium. The landlord requests a reservation and the liquidator validates it, checking that the day and time are available. Thus, as a result of this work, a *web* system was developed that allows the liquidator to have a more effective control of the condominium reservations. Among the technologies used in the development of the *web* application are Java language and JavaServer Faces.

Keywords: Condominiums. Ballroom. JavaServer Faces.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

AJAX	*Asynchronous Javascript and XML*
HTML	*HyperText Markup Language*
HTTP	*HyperText Protocol Transfer*
JSF	*JavaServer Faces*
MVC	*Model, View, Controller*
RF	Requisitos Funcionais
RIA	*Rich Internet Applications*
RNF	Requisitos Não Funcionais
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. No final do capítulo está a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Um condomínio é um espaço comum destinado ao uso de uma área habitacional ou comercial e que possui um espaço de uso individual e comum que pertencem aos diversos proprietários (MENDES; MARTINS, 2015).

A administração de um condomínio envolve diversos tipos de atividades, como controle financeiro, processos administrativos, controle de estoque, manutenções, gestão de reservas de áreas comuns, entre outras. Normalmente a administração é realizada pelo síndico, que é o representante legal do condomínio. O síndico pode ser um morador ou um profissional especializado em gestão condominial.

Para Farber (2005) existem três formas de gestão condominial, que são: autogestão, cogestão e terceirização. Na autogestão o síndico é o principal responsável pela gestão, a cogestão ocorre quando o síndico é assessorado por empresas especializadas na área e a terceirização ocorre quando uma empresa assume a responsabilidade da gestão (FARBER, 2005).

Quando ocorre a autogestão, normalmente são utilizadas planilhas eletrônicas, documentos e anotações para o gerenciamento dos processos administrativos e financeiros, ou seja, geralmente não há o auxílio de sistemas computacionais que automatizem as atividades de gestão.

Dentre as diversas atividades a serem gerenciadas em um condomínio está o controle de reserva de áreas comuns, pois a maioria dos condomínios dispõe dessa área (que pode ser mais de um ambiente e com reservas distintas) para que os usuários possam organizar reuniões, festas ou confraternizações.

Com base nesse contexto, este trabalho propõe o desenvolvimento de um sistema *web* que visa auxiliar o condômino na reserva de salão de festas em condomínios por meio de uma agenda *on-line* com as datas disponíveis para a reserva e para o síndico gerenciar esse espaço de forma mais efetiva.

1.2 OBJETIVOS

Os objetivos revelam o propósito de realização deste trabalho mediante os resultados esperados. O propósito principal se materializa no objetivo geral e os específicos são demarcados por ações mais detalhadas das metas que se pretende alcançar com o desenvolvimento do sistema proposto.

1.2.1 Geral

Desenvolver uma aplicação *web* para reserva de salão de festas em condomínios.

1.2.2 Específicos

- Possibilitar que condôminos possam reservar o salão de festas de seu condomínio de forma *on-line*.
- Possibilitar que o síndico tenha um controle mais efetivo das reservas do salão de festas um condomínio.
- Possibilitar o registro das reservas de um condomínio.

1.3 JUSTIFICATIVA

Um condomínio é representado por unidades autônomas que são identificadas e com áreas de uso comum pelos diversos proprietários. A relação interna do condomínio entre os moradores é pautada na convenção condominial que estabelece normas derivadas da vontade das partes interessadas. Além disso, há o regimento interno que pode ser elaborado por meio de convenções, na quais são estabelecidos os direitos e os deveres dos condôminos.

Em condomínios residenciais é comum haver áreas de uso comum, como o salão de festas, que é utilizado para eventos e confraternizações entre amigos e/ou família pelos condôminos. Essas áreas comuns devem ser reservadas seguindo as normas estabelecidas e de acordo com o regulamento interno. As reservas são solicitadas para o síndico que deve autorizar e agendar o uso do salão de festas conforme solicitação do condômino.

É comum os condôminos realizam a reserva por meio de contato telefônico, e-mail ou pessoalmente. Após a solicitação da reserva, o síndico deve agendar data e horário e isso normalmente é realizado por planilhas eletrônicas, planilhas impressas ou até mesmo agendas *on-line* como o Google Agenda. No entanto, as informações podem ficar dispersas se não

forem devidamente organizadas em um único ambiente, como, por exemplo, um banco de dados. E, isso pode gerar problemas como, por exemplo, duas ou mais reservas efetivadas no mesmo dia e horário, solicitações não atendidas, produtos ou equipamentos danificados ou extraviados sem um responsável vinculado, entre outros.

A justificativa de realização desse trabalho está alinhada com esse contexto, pois propõe um sistema para auxiliar os condôminos e o síndico nas reservas do salão de festas. O condômino tem mais flexibilidade em fazer a sua reserva, pois poderá acessar o sistema e cadastrar a reserva e aguardar que o síndico faça a validação da reserva. Dessa forma, o sistema permite uma comunicação mais efetiva entre o condômino e o síndico e um maior controle do uso do ambiente, incluindo possíveis danos a utensílios e móveis do local.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos dos quais este é o primeiro e apresenta as considerações iniciais, os objetivos e a justificativa do trabalho. No Capítulo 2 são apresentados os materiais utilizados para o desenvolvimento do trabalho. No Capítulo 3 está o resultado da realização do trabalho, que é composta pela modelagem e explanação do desenvolvimento do sistema sendo feita por meio do escopo, detalhamento de requisitos, modelagem do banco, explicações textuais, apresentação de telas e listagens de código. O Capítulo 4 apresenta a conclusão do trabalho desenvolvido e, por fim, estão as referências utilizadas no texto.

2 MATERIAIS E PROCEDIMENTOS

Esse capítulo tem por objetivo elencar os materiais utilizados na modelagem e na implementação do aplicativo desenvolvido como resultado da realização deste trabalho.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo.

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

Ferramenta / Tecnologia	Versão	Disponível em:	Aplicação
Admin-template	1.0.2	https://adminfaces.github.io/site/	*Framework front-end*
Astah Community	7.2.0	http://astah.net/editions/community	Modelagem do sistema: desenvolvimento do diagrama de casos de uso
Bootstrap	4.3v	https://getbootstrap.com/	*Framework front-end*
Java	1.8.0_221	https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html	Linguagem de programação
JavaServer Faces	2.3	https://javaee.github.io/javaserverfaces-spec/	Especificação Java para a construção de interfaces
Joinfaces	3.2.6	http://joinfaces.org/	*Framework* para criação de aplicações
pgAdmin 4	v4.14	https://www.pgadmin.org/download/	Ferramenta para administração do banco de dados.
PostgreSQL	10	https://www.postgresql.org/	Banco de dados
PrimeFaces	7.0	https://www.primefaces.org/downloads/	*Framework front-end*
Spring Boot	2.0.6.RELEASE	https://projects.spring.io/spring-boot/	*Framework* para criação de aplicações
Spring Data	2.0.11.RELEASE	https://projects.spring.io/spring-data-jpa/	*Framework* de implementação de repositórios
Spring E-mail	2.0.1.RELEASE	https://projects.spring.io/spring-boot/	*Framework* para envio de *e-mail*
Spring Security	5.0.9.RELEASE	https://projects.spring.io/spring-security/	*Framework* de autenticação e segurança da aplicação.
Spring Tool Suite 4	4.4.0.RELEASE	https://spring.io/tools	Ambiente de desenvolvimento para o sistema

Fonte: Autoria própria (2019).

2.1.1 Java Server Faces

O *JavaServer Faces* (JSF) é uma especificação para criar interfaces gráficas de aplicações *web*. É um padrão (*standard*) para Java EE. A ideia base é, após alguns anos de evolução (esta especificação nasceu em 2004), sistematizar o que as interfaces *web* têm em comum. Com relativa facilidade permite desenvolver interfaces *web* ricas e interativas (cujas aplicações são denominadas *Rich Internet Application* - RIA). É *server-side* e totalmente orientada a componente ("pedaço" reutilizável da interface). Contudo, além de uma série de componentes (abstrações dos componentes *HyperText Markup Language* - HTML), prevê validadores, gestão de estado, conversores, modelo de eventos e outros.

Além disso, o JSF considera capacidades de internacionalização, *templating*, acessibilidade, navegação, *Asynchronous Javascript and XML* (AJAX), entre outras. É de alto nível, pelo que o modelo *HyperText Protocol Transfer* (http), *servlets* e outros conceitos da página são implícitos.

Sendo o JSF uma especificação, as suas duas implementações mais comuns (*frameworks*) são o Apache MyFaces e o Oracle Mojarra. Tanto o JSF como o Struts pertencem ao mundo Java e ambos apoiam o desenvolvimento de interfaces gráficas *web*. A maior diferença é que o JSF foi pensado para aplicações e o Struts para sites. Apesar do JSF servir para *web*sites, esse não é o seu uso mais comum (exceto quando se tratam de sites muito orientados à tarefa, com necessidades de componentes ricos e variados e diversas interações AJAX) (SOARES, 2013).

2.1.2 PrimeFaces

PrimeFaces é uma biblioteca de componentes ricos em JSF para o desenvolvimento de interface gráfica para a *web*. PrimeFaces é uma biblioteca de componentes de código aberto para o JSF, permitindo criar interfaces ricas para aplicações *web* de forma simplificada e eficiente. PrimeFaces possui um rico conjunto de componentes de interface (DataTable, AutoComplete, HTMLEditor, gráficos, etc.). Nenhum *Extensible Markup Language* (XML) de configuração extra é necessário e não há dependências.

Os componentes do PrimeFaces foram construídos para trabalhar com AJAX por "*default*", isto é, não é necessário nenhum esforço por parte do desenvolvedor para realização de chamadas assíncronas ao servidor. Além disso, o PrimeFaces permite a aplicação de temas

(*skins*) com o objetivo de mudar a aparência dos componentes de forma simples (GAMERS, 2012).

2.1.3 JOINFACE

JoinFace é projeto que permite o uso de JSF no Spring Boot. Ele configura automaticamentes bibliotecas PrimeFaces, AdminFaces, BootsFaces, ButterFaces, IceFaces, RichFaces, OmniFaces, AngularFaces, Mojarra e MyFaces para executar em contêineres de servlet Tomcat, Jetty ou Undertow incorporados. Ele também configura automaticamente o Weld e Rewrite. Além de visar solucionar os recursos de integração JSF e Spring Boot. A versão atual inclui suporte a anotações JSF e CDI e suporte a Spring Security JSF Facelet Tag (JOINFACES, 2019).

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema proposto.

3.1 ESCOPO DO SISTEMA

O sistema para realizar a reserva de salão de festas de um condomínio visa proporcionar aos usuários maior controle e comodidade no momento de efetuar uma reserva, pois é necessário apenas acessar o sistema e cadastrar a sua reserva. Os cadastros necessários para o controle de reservas são o de usuários (síndico e condôminos) e o de reservas.

Ao acessar o sistema pela primeira vez para efetuar uma reserva, o condômino deverá realizar seu próprio cadastro para posterior validação. Na reserva deve ser informada a data e horário, quantidade de convidados, tipo de evento (festa de aniversário, almoço ou jantar, por exemplo). Por meio da agenda do sistema de reservas, o condômino poderá visualizar as datas que estão disponíveis em forma de calendário e registrar uma nova reserva.

Ao efetuar a reserva, o condômino deverá concordar ou não com o termo de responsabilidade, alegando estar ciente dos cuidados com o uso do salão de festas por meio da assinatura abaixo no *link* do termo de responsabilidade.

Ao realizar a reserva, o condômino deve aguardar que o síndico faça a validação da reserva. Isso acontece por meio do *status* da reserva. Ao realizar a reserva, o *status* fica registrado como “pendente”. Após validar a reserva o síndico altera o *status* para “reservado” ou “indeferido”. No caso de a reserva ser indeferida será registrada o motivo para que o condômino tenha ciência. Após a alteração do *status* o condômino e o síndico recebem um e-mail informando o status da sua reserva.

3.2 MODELAGEM DO SISTEMA

Esta seção apresenta os requisitos funcionais e não funcionais e o diagrama de casos de uso que representa os atores do sistema e as funcionalidades definidas para cada ator e o diagrama de entidades e relacionamento do banco de dados.

No Quadro 2 estão os requisitos funcionais (RF) definidos para o sistema.

Quadro 2 – Requisitos Funcionais

Identificação	Nome	Descrição
RF1	Manter usuário	As ações no sistema são realizadas por dois usuários distintos: o síndico que é o ator responsável por todas as funcionalidades do sistema e o condômino que realiza as operações de cadastro próprio e de reserva do salão de festas. Essas ações do condômino são validadas pelo síndico que exerce a função de administrador do sistema.
RF2	Validar usuário	Quando o condômino realiza seu cadastro, ele informa a sua unidade habitacional (número do apartamento). A validação do cadastro do usuário é realizada quando o síndico torna o cadastro do usuário ativo por meio da unidade habitacional.
RF3	Manter reserva	Quando uma reserva é realizada o sistema exibe o cadastro de reserva, cujos dados pessoais como nome e unidade habitacional são preenchidos automaticamente conforme o usuário autenticado no sistema. O usuário deverá informar os campos de tipo de evento, data inicial e final do evento e quantidade de pessoas que participarão do evento. Além disso, o usuário poderá fazer o *download* do documento com os termos e condições de uso do salão de festas e assinar digitalmente a reserva, alegando estar ciente das regras de uso do salão de festas.
RF4	Manter *status* da reserva	Quando uma reserva é solicitada o *status* é automaticamente atualizado para "pendente". Após o síndico realizar os procedimentos necessários para a reserva (conferência de produtos, equipamentos, etc.), o *status* será atualizado para "reservado" ou "indeferido" e, nesse caso, será registrado o motivo do indeferimento.
RF5	Validar reserva	O síndico deverá validar a reserva realizada pelo condômino. Para isso, deverá acessar a tela de reservas e atualizar o *status* da reserva para "reservado" no caso de a reserva ser procedente, "indeferido" caso a reserva não seja possível e, nesse caso, deverá informar o motivo para não autorizar a reserva. Quando uma reserva á cadastrada o *status* fica, por padrão, como "pendente".
RF06	Manter unidade de moradia	Registro das unidades de moradia, apartamentos, por exemplo, que pertencem ao condomínio.

Fonte: Autoria própria (2019).

No Quadro 3 estão os requisitos não funcionais (RNF) do sistema e estão relacionados ao acesso ao sistema, validação de campos e de relatórios.

Quadro 3 – Requisitos não funcionais

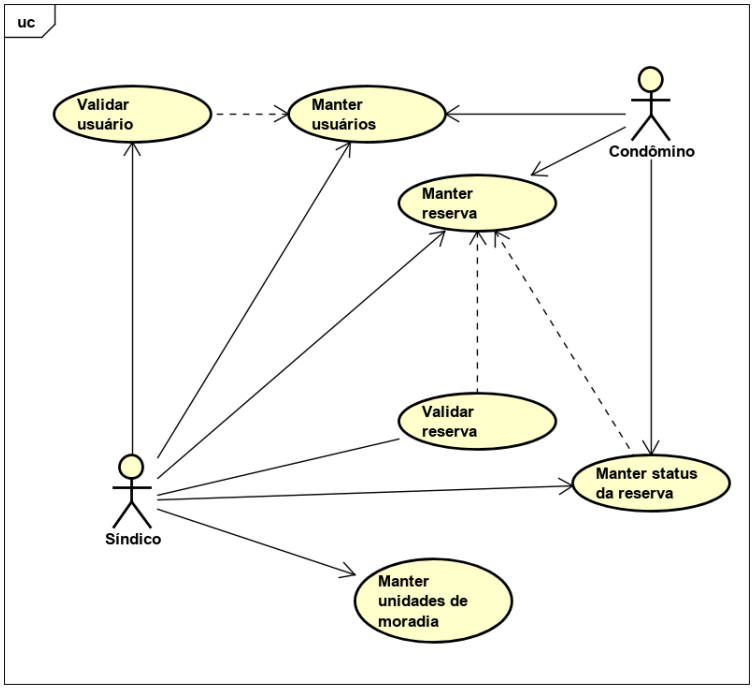
Identificação	**Nome**	**Descrição**
RNF01	Efetuar *login*	O acesso ao sistema será realizado por meio de *login* e senha.
RNF02	Campos de preenchimento obrigatório	Os campos que são de preenchimento obrigatório são validados por meio de funções do sistema.
RNF03	Campos com máscara de entrada	Os campos que possuem caracteres especiais, e que não serão armazenados no banco de dados, serão validados por meio de máscaras de entrada nos respectivos campos.
RNF04	Vínculo entre dados	O sistema também não deve permitir a exclusão de dados vinculados. Por exemplo: excluir uma categoria que está vinculada a um cadastro de serviços.
RNF05	Autenticação de usuários	O usuário somente poderá autenticar-se no sistema após o seu cadastro ter sido validado.
RNF06	Realizar reserva	Uma reserva só pode ser registrada com, no mínimo, 72h de antecedência.
RNF07	Exclusão de uma reserva	Uma reserva não poderá ser excluída antes de 72h antes do evento.
RNF08	Data final não pode	A data final da reserva não pode ser menor que a data inicial da

	ser menor que data inicial	reserva.

Fonte: Autoria própria.

O diagrama de casos de uso apresentado na Figura 1 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: síndico e condômino. O condômino realiza seu próprio cadastro que deve ser validado pelo síndico. Com o cadastro validado o condômino pode realizar suas reservas preenchendo os campos correspondentes aos dados da reserva. O tipo de evento (festa de aniversário, almoço, jantar, por exemplo) representa uma lista de valores pré-definidos implementada no sistema (*enum*). Porém, ao solicitar uma reserva o condômino poderá informar um tipo que não esteja na lista padrão por meio de um campo de texto.

Figura 1 – Diagrama de casos de uso



Fonte: Autoria própria (2019).

Os quadros 4 a 7 apresentam a descrição dos casos de uso que envolvem as operações de inclusão, alteração, exclusão e consulta. Esses casos de usos identificados como “manter” no diagrama de casos de uso apresentado na Figura 1.

Quadro 4 - Operação de incluir dos casos de uso de cadastro

Caso de uso: Incluir (refere-se à operação de inclusão de todos os casos de usos identificados como "manter"). **Descrição:** Inclusão dos dados cadastrais no sistema. **Evento Iniciador:** Ator solicita inclusão de um registro no sistema. **Atores:** Síndico ou condômino, de acordo com suas funções definidas no caso de uso. **Pré-condição:** Não há. **Sequência de Eventos:** 1. Ator acessa a tela para realizar o cadastro inserindo as informações necessárias. 2. O sistema insere as informações no banco de dados e informa ao usuário o status do procedimento. **Pós-Condição:** Registro inserido no banco de dados.	
Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. Ator deixa de informar os dados obrigatórios. 1.2. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.
2. Campos informados em formato incorreto.	2.1. O ator informa dados em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.

Fonte: Autoria própria (2019).

Quadro 5 - Operação de alterar dos casos de uso de cadastro

Caso de uso: Alterar (refere-se à operação de alteração de todos os casos de usos identificados como "manter"). **Descrição:** Alteração dos dados cadastrais no sistema. **Evento Iniciador:** Ator solicita alteração de um registro no sistema. **Atores:** Síndico ou condômino de acordo com suas funções definidas no caso de uso. **Pré-condição:** Registro estar incluso no sistema. **Sequência de Eventos:** 1. Ator acessa a tela para visualização dos dados do registro. 2. O sistema apresenta o registro selecionado para alteração. 3. Ator altera os dados do registro. 4. O sistema altera as informações no banco de dados e informa ao usuário o status

do procedimento. **Pós-Condição:** Registro alterado no banco de dados.	
Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. Ator exclui dados obrigatórios. 1.2. O sistema permanece na tela de edição mantendo as alterações realizadas.
2. Campos informados em formato incorreto.	2.1. Ator altera os dados deixando em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de edição mantendo as alterações realizadas.

Fonte: Autoria própria (2019).

Quadro 6 - Operação de excluir dos casos de uso de cadastro

Caso de uso: Excluir (refere-se à operação de exclusão de todos os casos de usos identificados como "manter"). **Descrição:** Exclusão dos dados cadastrais no sistema. **Evento Iniciador:** Ator solicita exclusão de um registro no sistema. **Atores:** Síndico ou condômino de acordo com suas funções definidas no caso de uso. **Pré-condição:** Registro estar incluso no sistema. **Sequência de Eventos:** 1. Ator acessa a tela para exclusão do registro. 2. O sistema exclui as informações no banco de dados e informa ao usuário o status do procedimento. **Pós-Condição:** Registro excluído no banco de dados. **Extensões:** Registro possuir vínculo com outros cadastros	
Nome do fluxo alternativo (extensão)	Descrição
1. Exclusão de registro com vínculos no sistema	1.1. Ator clica em excluir um registro que possui vínculos no sistema. 1.2. O sistema verifica que o registro tem vínculos, não exclui o mesmo e exibe mensagem de alerta ao usuário.

Fonte: Autoria própria (2019).

Quadro 7 - Operação de buscar dos casos de uso de cadastro

Caso de uso: Consultar (refere-se à operação de consulta de todos os casos de usos identificados como "manter").

Descrição:
Consulta dos dados cadastrais dos registros do sistema.
Evento Iniciador:
Ator solicita consulta de um registro no sistema.
Atores:
Administrador ou Cliente de acordo com suas funções definidas no caso de uso.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O ator indica os filtros desejados para consulta.
3. O sistema apresenta os dados da consulta ao usuário.

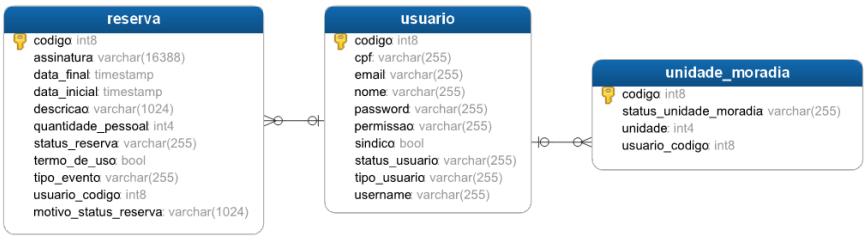
Pós-Condição:
Dados da consulta apresentados ao usuário.

Fonte: Autoria própria (2019).

3.2.1 Diagrama de Entidade e Relacionamento (DER)

Na Figura 2 são exibidos os relacionamentos entre as entidades do sistema, os tipos e os tamanhos de cada atributo.

Figura 2 - Diagrama de entidade e relacionamento



Fonte: Autoria própria (2019).

3.3 APRESENTAÇÃO DO SISTEMA

Esta seção apresenta a descrição das principais funcionalidades que poderão ser visualizadas por meio de *prints* das telas do sistema.

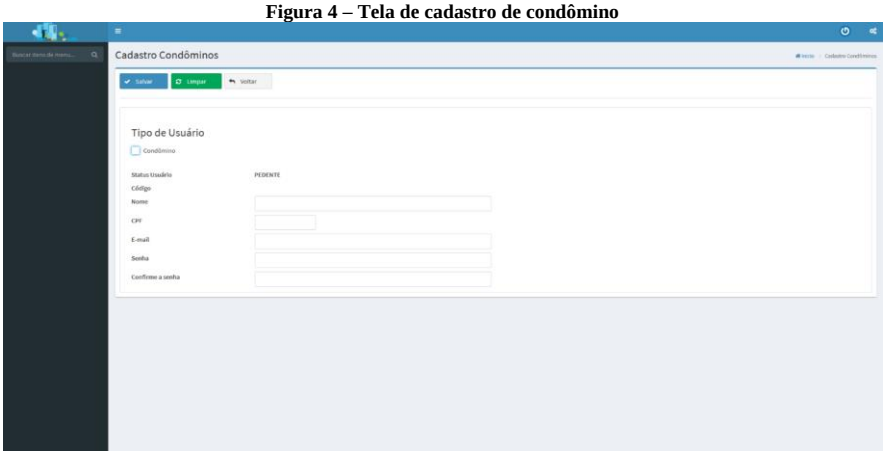
A Figura 3 representa a tela de autenticação de usuário, que pode ser o síndico ou o condômino. Para isso, é necessário informar os dados de usuário e senha. Se o usuário clicar no botão “Entrar” para efetuar o *login* e não informar os dados corretamente é apresentada uma mensagem de erro.

Figura 3 – Tela de autenticação de usuário

Fonte: Autoria própria (2019).

Na tela apresentada na Figura 3, também é possível que o usuário faça o seu cadastro por meio da opção “Cadastrar Novo Condômino” (da tela de cadastro da Figura 3). Ao clicar nesse botão o usuário é redirecionado para a tela de cadastro de condômino apresentada na Figura 4.

Figura 4 – Tela de cadastro de condômino



Fonte: Autoria própria (2019).

Na tela apresentada na Figura 4, o usuário faz o seu cadastro por meio do preenchimento dos campos da tela com as informações pessoais. Ao clicar na opção "condômino" ativará um campo para colocar o número da unidade habitacional.

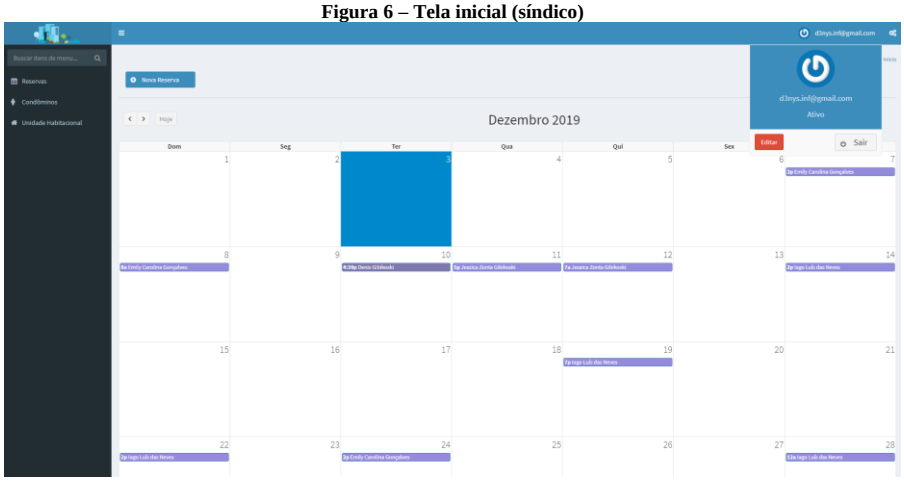
O condômino poderá redefinir sua senha ao clicar na opção "Redefinir senha" que está na tela de *login* apresentada na Figura 3. Ao clicar nessa opção, ele deverá informar seu *e-mail* e clicar no botão "Redefinir senha", conforme apresentado na Figura 5.

Figura 5 – Tela de redefinir senha

Fonte: Autoria própria (2019).

A tela apresentada na Figura 6 é a que possui as permissões do síndico. Essa tela apresenta a agenda com as reservas no mês e um menu do lado esquerdo que permite que o síndico faça validações de condômino, das unidades habitacionais e das reservas. No cabeçalho do sistema estão os botões para realizar o cadastro de uma reserva e o *e-mail* do usuário autenticado. Para sair do sistema é necessário clicar sobre o *e-mail* e, em seguida, clicar no botão "Sair". Para editar os dados do cadastro (do síndico ou do condômino) é necessário clicar no botão "Editar", conforme apresentado na Figura 6.

Figura 6 – Tela inicial (síndico)



Fonte: Autoria própria (2019).

Quando o condômino realiza a autenticação no sistema, uma tela semelhante à da Figura 6 é apresentada, porém será mostrado somente o menu para que ele possa cadastrar uma reserva.

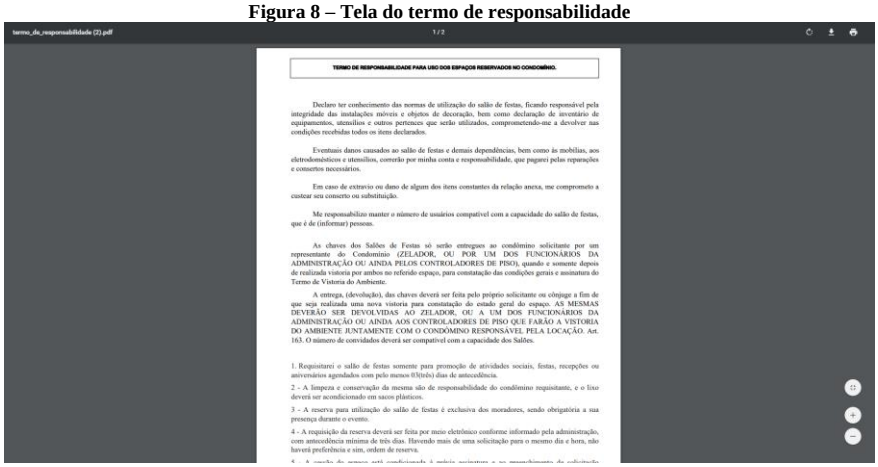
Ao clicar no botão “Nova Reserva” será aberta a tela apresentada na Figura 7. Nessa tela é possível cadastrar uma reserva. Os campos nome e unidade habitacional são automaticamente preenchidos conforme o usuário autenticado no sistema. Os dados como data de início e término da reserva, quantidade de convidados e tipo de evento que será realizado (festa de aniversário, almoço ou janta, por exemplo) deverão ser informados pelo condômino. Além disso, ele deverá concordar com os termos e condições de uso do salão de festas por meio de assinatura digital, conforme apresentado na Figura 7.

Figura 7 – Tela de cadastro de reserva

Fonte: Autoria própria (2019).

A Figura 8 apresenta o termo de responsabilidade para uso do condômino. O condômino terá acesso a esse documento após clicar no *link* "termo" apresentado na Figura 7.

Figura 8 – Tela do termo de responsabilidade



Fonte: Autoria própria (2019).

A Figura 9 mostra a lista de reservas que é acessada quando o síndico clica na opção "Reservas". Nesta tela há o botão "Nova reserva" que permite criar uma nova reserva, editar e excluir uma reserva.

Figura 9 – Listagem de reserva

Fonte: Autoria própria (2019).

Para validar uma reserva o síndico deve clicar no botão “editar” (em cor azul na tela da Figura 9) e alterar o status da reserva por meio da opção “status evento” da Figura 10.

Figura 10 – Tela para validar reserva

Fonte: Autoria própria (2019).

A listagem de condômino é apresentada na Figura 11. Para validar um cadastro de condômino, o síndico também deverá clicar no botão “editar”. Por meio dessa tela também é possível cadastrar um novo condômino e excluir.

Figura 11 – Tela de lista de condômino

Fonte: Autoria própria (2019).

Na tela apresentada na Figura 12, há a lista de unidade habitacional na qual haverá a validação das unidades mudando os *status* da unidade para “ativo”, “inativo” ou “pendente”. Quando o status for definido para “inativo” são será possível excluí-lo.

Figura 12 – Tela de lista de unidade habitacional

Fonte: Autoria própria (2019).

3.4 IMPLEMENTAÇÃO DO SISTEMA

O primeiro passo do desenvolvimento do sistema foi a estruturação do banco de dados da aplicação, com seu conjunto de entidades, procedimentos e rotinas.

A codificação do sistema foi feita em linguagem Java, no ambiente de desenvolvimento integrado do Spring Tool Suite 4, e foi utilizando um projeto Maven chamado de "AdminFaces", que é um projeto de código aberto que integra Primefaces, Bootstrap e Admin LTE para criar aplicativos JSF totalmente responsivos. Este projeto possui vários *starters* como se fosse o Spring. Nesse trabalho foi utilizado o starter "Admin Starter Spring Boot Security", este projeto e aplicativo Admin Starter usando SpringBoot, JoinFaces e Spring Security.

JoinFaces permite o uso de JSF no aplicativo Spring Boot. Ele configura automaticamente as bibliotecas PrimeFaces, AdminFaces e Mojarra para executar em contêineres de *servlet* Tomcat incorporados. Ele também configura automaticamente o Weld e visa solucionar os recursos de integração JSF e Spring Boot. A versão atual inclui suporte a anotações JSF e CDI e suporte a Spring Security JSF Facelet Tag.

A Listagem 1 exibe o start Spring Boot do projeto habita as execuções em assíncronas e a exclusão da classe padrão que envia *e-mail*.

Listagem 1 – Start do projeto Spring Boot

```
...
10 @EnableAsync
11 @SpringBootApplication
12 @EnableAutoConfiguration(exclude = MailSenderValidatorAutoConfiguration.class)
13 public class AdminBootMain {
14
15      public static void main(String[] args) {
16          SpringApplication.run(AdminBootMain.class, args);
17      }
18
19 }
```

Fonte: Autoria própria (2019).

No arquivo application.properties mostrado na Listagem 2 encontra-se as configurações de *e-mail*, do banco de dados com PostgreSQL e a ativação e configuração do *pool* de conexão utilizando Hikari.

Listagem 2 – Configuração application.properties

```
1 #Port HTML
 2 server.port=8081
```

```
 3
 4 #Configuracao de email
 5 spring.mail.host=smtp.gmail.com
 6 spring.mail.port=587
 7 spring.mail.username=softcondtcc@gmail.com
 8 spring.mail.password=######
 9 spring.mail.properties.mail.smtp.auth=true
10 spring.mail.properties.mail.smtp.starttls.enable=true
11 spring.mail.properties.mail.smtp.starttls.required=true
12 spring.mail.properties.mail.smtp.ssl.enable=false
13 spring.mail.test-connection=true
14
15 #Configuracao
16 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
17 spring.jpa.generate-ddl=true
18 spring.jpa.show-sql=true
19 spring.jpa.hibernate.ddl-auto=update
20
21 #Desativar a cria��o de LOB contextual como o m�todo createClob ()
gerou erro
22 spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false
23
24
25 # PostgreSQL
26 spring.datasource.url=jdbc:postgresql://localhost:5432/softcond
27 spring.datasource.username=postgres
28 spring.datasource.password=postgres
29 spring.datasource.driverClassName=org.postgresql.Driver
30
31 #Configuracao de Pool de Conexao
32 spring.datasource.type=com.zaxxer.hikari.HikariDataSource
33 spring.datasource.hikari.connection-timeout=20000
34 spring.datasource.hikari.minimum-idle=10
35 spring.datasource.hikari.maximum-pool-size=40
36 spring.datasource.hikari.idle-timeout=10000
37 spring.datasource.hikari.max-lifetime=1000
38 spring.datasource.hikari.auto-commit=true
39
40
```

Fonte: Autoria própria (2019).

A configuração do projeto o pom.xml que inicia as configurações e faz os *downloads* de todas as dependências do projeto. O pom.xml conta com as configurações dos *starters* do JoinsFaces e Spring Boot, conforme exibido na Listagem3.

Listagem 3 – arquivo pom.xml

```
 1 <?xml version="1.0"?>
 2 <project
 3         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
 4         xmlns="http://maven.apache.org/POM/4.0.0"
 5         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 6         <modelVersion>4.0.0</modelVersion>
 7         <groupId>com.softCond.adminfaces</groupId>
 8         <artifactId>SoftCond-starter</artifactId>
```

```
 9         <version>0.1-SNAPSHOT</version>
10         <packaging>jar</packaging>
11         <name>SoftCond</name>
12
13         <parent>
14                 <groupId>org.joinfaces</groupId>
15                 <artifactId>joinfaces-parent</artifactId>
16                 <version>3.2.6</version>
17                 <relativePath /> <!-- lookup parent from repository -
->
18         </parent>
19
20         <properties>
21   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
22                 <mojarra.version>2.3.4</mojarra.version>
23         </properties>
24
25         <dependencyManagement>
26                 <dependencies>
27                         <dependency>
28                     <groupId>com.github.adminfaces</groupId>
29                     <artifactId>admin-template</artifactId>
30                                 <version>1.0.2</version>
31                         </dependency>
32                         <dependency>
33                            <groupId>com.github.adminfaces</groupId>
34                                 <artifactId>admin-theme</artifactId>
35                                 <version>1.0.2</version>
36                         </dependency>
37                         <dependency>
38                                 <groupId>org.primefaces</groupId>
39                                 <artifactId>primefaces</artifactId>
40                                 <version>7.0</version>
41                         </dependency>
42                         <dependency>
43                <groupId>org.primefaces.extensions</groupId>
44               <artifactId>primefaces-extensions</artifactId>
45                            <version>7.0.2</version>
46                         </dependency>
47                 </dependencies>
48         </dependencyManagement>
49
50         <dependencies>
51                 <!-- POSTGRESQL -->
52                 <dependency>
53                         <groupId>org.postgresql</groupId>
54                         <artifactId>postgresql</artifactId>
55                         <scope>compile</scope>
56                 </dependency>
57
58                 <!-- Spring DATA -->
59                 <dependency>
60                         <groupId>org.springframework.boot</groupId>
61                 <artifactId>spring-boot-starter-data-jpa</artifactId>
62                 </dependency>
63
64                 <!-- Spring -->
65                 <dependency>
66                         <groupId>org.springframework.boot</groupId>
67                   <artifactId>spring-boot-starter-mail</artifactId>
68                 </dependency>
```

```
 69
 70                 <!-- SPRING BOOT -->
 71                 <dependency>
 72               <groupId>org.springframework.boot</groupId>
 73                 <artifactId>spring-boot-starter-web</artifactId>
 74                 </dependency>
 75
 76                 <dependency>
 77                         <groupId>org.springframework.boot</groupId>
 78                         <artifactId>spring-boot-starter</artifactId>
 79                 </dependency>
 80                 <dependency>
 81                 <groupId>org.springframework.boot</groupId>
 82                 <artifactId>spring-boot-starter-security</artifactId>
 83                 </dependency>
 84                 <dependency>
 85                         <groupId>org.springframework.boot</groupId>
 86                  <artifactId>spring-boot-starter-test</artifactId>
 87                         <scope>test</scope>
 88                 </dependency>
 89                 <dependency>
 90                         <groupId>org.springframework.boot</groupId>
 91                         <artifactId>spring-boot-devtools</artifactId>
 92                 </dependency>
 93                 <dependency>
 94                         <groupId>org.joinfaces</groupId>
 95               <artifactId>adminfaces-spring-boot-starter</artifactId>
 96                 </dependency>
 97                 <dependency>
 98                         <groupId>jstl</groupId>
 99                         <artifactId>jstl</artifactId>
100                         <version>1.2</version>
101                 </dependency>
102         </dependencies>
103         <build>
104                 <finalName>SoftCond</finalName>
105
106                 <plugins>
107                         <plugin>
108             <groupId>org.springframework.boot</groupId>
109            <artifactId>spring-boot-maven-plugin</artifactId>
110                                 <configuration>
111                                         <source>1.8</source>
112                                         <target>1.8</target>
113                                 </configuration>
114                         </plugin>
115                 </plugins>
116         </build>
117 </project>
118
```

Fonte: Autoria própria (2019).

O projeto JoinFaces traz o *template* do *front-end* configurado que está sendo mostrado na Listagem 4.

Listagem 4 – template.xhtml

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
 3          xmlns:ui="http://java.sun.com/jsf/facelets"
 4          xmlns:f="http://java.sun.com/jsf/core"
```

```
 5         xmlns:h="http://java.sun.com/jsf/html"
template="/admin.xhtml">
 6
 7         <ui:define name="head">
 8                 <title>Soft Cond</title>
 9                 <link rel="shortcut icon" type="image/x-icon"
10         href="#{resource['favicon/favicon.ico']}" />
11                 <link rel="shortcut icon" type="image/x-icon"
12         href="#{resource['favicon/favicon.ico']}" sizes="16x16"
/>
13                 <link rel="shortcut icon" type="image/x-icon"
14         href="#{resource['favicon/favicon.ico']}" sizes="32x32"
/>
15                 <link rel="shortcut icon" type="image/x-icon"
16        href="#{resource['favicon/favicon.ico']}" sizes="96x96"
/>
17                 <link rel="manifest" href="manifest.json" />
18                 <meta name="theme-color" content="#444" />
19                 <meta name="mobile-web-app-capable"
content="yes" />
20                 <f:loadBundle var="msg" basename="starter" />
21                 <h:outputStylesheet library="css"
name="starter.css" />
22         </ui:define>
23
24         <ui:define name="logo">
25    <h:graphicImage library="images" name="EMPRESA-SALUDABLE-
ex2.png" />
26         </ui:define>
27
28         <ui:define name="logo-mini">
29     <h:graphicImage library="images" name="746859-32x32.png" />
30         </ui:define>
31
32         <ui:define name="menu">
33                 <ui:include src="/includes/menu.xhtml" />
34         </ui:define>
35
36         <ui:define name="top-menu">
37                 <ui:include src="/includes/top-bar.xhtml" />
38         </ui:define>
39
40
41 </ui:composition>
```

Fonte: Autoria própria (2019).

A segurança foi implementada utilizando o Spring Security. Para fazer a implementação foi criado a classe SecurityConfig, apresentada na Listagem 5. Esta classe configura as autorizações e as autenticações do sistema, sendo que esta configuração é um padrão *Model*, *View*, *Controller* (MVC), em que a cada requisição verificará se há um usuário autenticado e verificará as permissões para cada usuário.

Listagem 5 – implementação Spring Config

```
...
12
13 @Configuration
14 @Enable WebSecurity
15 public class SecurityConfig extends WebSecurityConfigurerAdapter {
16
17         @Autowired
18         private AppUserDetailsService userDetailsService;
19
20         @Bean
21         public PasswordEncoder passwordEncoder() {
22                 return new BCryptPasswordEncoder();
23         }
24
25         @Autowired
26     public void configureGlobal(AuthenticationManagerBuilder auth)
27                 throws Exception {
28         auth.userDetailsService(userDetailsService)
29                 .passwordEncoder(passwordEncoder());
30         }
31
32         @Override
33         protected void configure(HttpSecurity http)
34                 throws Exception {
35
36         // form login
37         http.headers().frameOptions().sameOrigin()
38         .and()
39         .csrf().disable()
40         .authorizeRequests()
41         .antMatchers("/cadastroUsuario.xhtml", "/login.xhtml",
42         "/resources/**", "/redefinirSenha.xhtml",
43         "/javax.faces.resource/**").permitAll()
44         .antMatchers("/index.xhtml","/listaEventos.xhtml",
45          "/cadastroEvento.xhtml").authenticated()
46         .antMatchers("/listaUsuarios.xhtml")
47                 .hasAnyRole("SINDICO")
48         .antMatchers("/listaUnidadeMoradia.xhtml")
49                 .hasAnyRole("SINDICO")
50         .anyRequest().authenticated()
51         .and().formLogin().loginPage("/login.xhtml")
52         .defaultSuccessUrl("/index.xhtml")
53         .failureUrl("/login.xhtml?authfailed=true")
54                 .permitAll()
55         .and().logout()
56         .logoutSuccessUrl("/login.xhtml")
57   .logoutUrl("/j_spring_security_logout")
58    .invalidateHttpSession(true).deleteCookies("JSESSIONID");
59         }
60 }
61
```

Fonte: Autoria própria (2019).

O código da tela inicial está representado na listagem 6. Para fazer a codificação deste projeto, para o gerenciamento do ciclo de vida dos *beans* usa-se anotações do CDI que vem acoplado no projeto *starters*. *P*or meio do método inicializar(), localizado na linha 47 da Listagem 6**,** inicializa o componente que simula um calendário para listar as reservar existente. E o método onEventSelect(), localizado na linha 75, abrirá uma caixa de diálogo com as informações referente à reserva selecionada.

Listagem 6 – HomeBean.java

```
...
27
28 @Named
 29 @ViewScoped
 30 public class HomeBean implements Serializable {
 31
 32         private static final long serialVersionUID = 1L;
 33
 34         @Autowired
 35         private ApplicationEventPublisher publisher;
 36
 37         @Autowired
 38         private ReservaService service;
 39         private Reserva reserva;
 40         private List<Reserva> listaReservas;
 41         private ScheduleModel reservas;
 42         private ScheduleEvent event;
 43         private Date hoje;
 44         private UsuarioSistema sistema;
 45
 46 @PostConstruct
 47 public void inicializar() {
 48     try {
 49         if
(!"anonymousUser".equals(SecurityContextHolder.getContext().getAuth
entication().getName())) {
 50         sistema = (UsuarioSistema)
SecurityContextHolder.getContext().getAuthentication().getPrincipal
();
 51         }
 52
 53         setHoje(new Date());
 54         reservas = new DefaultScheduleModel();
 55         reserva = new Reserva();
 56         if (sistema != null) {
 57             reserva.setUsuario(sistema.getUsuario());
 58         }
 59         listaReservas = service.todos();
 60
 61         for (Reserva reserva : listaReservas) {
 62       DefaultScheduleEvent evt = new DefaultScheduleEvent();
 63
```

```
evt.setStyleClass(ScheduleUtil.getClass(reserva.getStatusReserva())
);
 64              evt.setEndDate(reserva.getDataFinal());
 65              evt.setStartDate(reserva.getDataInicial());
 66              evt.setDescription(reserva.getDescricao());
 67              evt.setTitle(reserva.getUsuario().getNome());
 68              reservas.addEvent(evt);
 69          }
 70          } catch (Exception e) {
 71     FacesUtil.addFatalMessage("Erro FATAL " + e.getMessage());
 72      }
 73 }
 74
 75 public void onEventSelect(SelectEvent selectEvent) {
 76      event = (DefaultScheduleEvent) selectEvent.getObject();
 77      listaReservas.forEach(e -> {
 78          if (event.getDescription().equals(e.getDescricao()) &&
 79              event.getTitle().equals(e.getUsuario().getNome())&&
 80              event.getStartDate().equals(e.getDataInicial()) &&
 81              event.getEndDate().equals(e.getDataFinal())) {
 82                  reserva = e;
 83          }
 84          });
 85          }
 86          public boolean habilitarDescricao() {
 87              return reserva.getDescricao() != null;
 88          }
 89          public void novoEvento() {
 90       FacesUtil.redirecionarPagina("cadastroEvento.xhtml");
 91          }
 92          public void redicionaCadastroUsuario() {
 93 FacesUtil.redirecionarPagina("cadastroUsuario.xhtml?usuario=" +
sistema.getUsuario().getCodigo());
 94          }
...
```

Fonte: Autoria própria (2019).

Na Listagem 7 são mostrados os métodos principais da classe que responsável pelo gerenciamento das reservas. Também conta com método para o usuário fazer o *dowloand* do termo de uso, localizado na linha 125 da Listagem 7. O sistema verifica se o usuário tem conhecimento do termo de responsabilidade por meio do componente de assinatura (exibida na Figura 7 que mostra a tela de cadastro de reserva). Após assinatura efetuada é acionado um evento AJAX que valida o termo.

Listagem 7 – CadastroEventoBean.java

```
...
 70
 71 public void removerEvento() {
 72        try {
```

```
 73         RestricaoHorario.permite(reserva.getDataFinal());
 74         service.remover(reserva);
 75         reserva.setStatusReserva(StatusReserva.EXCLUIDO);
 76         publisher.publishEvent(reserva);
 77         inicializar();
 78        esUtil.addInfoMessage("Reserva Excluida com sucesso!!!");
 79        } catch (Exception e) {
 80        FacesUtil.addFatalMessage(e.getMessage());
 81       }
 82  }
...
 88 public void novoEvento() {
 89  try {
 90       if (isTermoResposabilidade(reserva.getTermoDeUso())) {
 91           service.salvar(reserva);
 92           publisher.publishEvent(reserva);
 93           FacesUtil.addInfoMessage("Evento " +
 94           reserva.getCodigo() + "  salvo !!");
 95           FacesUtil.addInfoMessage("Unidade N°:  " +
 96  reserva.getUsuario().getMoradia().getUnidade() + "  salvo !!");
 97      }
 98   } catch (Exception e) {
 99     FacesUtil.addErrorMessage("ERRO :" + e.getMessage());
100           }
101  }
102
103   private boolean isTermoResposabilidade(Boolean termoDeUso) {
104                if (termoDeUso) {
105                        return true;
106                } else {
107 throw new NegocioException("O Termo de Uso não foi Assinado");
108                }
109        }
110
...
115        public void ativaTermo() {
116                try {
117                        if (!reserva.getAssinatura().isEmpty())
{
118                                reserva.setTermoDeUso(true);
119                        }
120                } catch (Exception e) {
121                        new NegocioException("Sem Assinatura");
122                }
123        }
124
125 public StreamedContent termoDeUso() {
126           InputStream stream =
FacesContext.getCurrentInstance().getExternalContext()
127
.getResourceAsStream("/resources/termo_de_responsabilidade.pdf");
128                DefaultStreamedContent file = new
DefaultStreamedContent(stream, "application/pdf",
129
```

```
"termo_de_responsabilidade.pdf");
130                 return file;
131         }
132
...
```

Fonte: Autoria própria (2019).

A implementação do serviço da reserva está apresentada na Listagem 8, nos métodos "remover" e "salvar". Nestes métodos verifica se há validações de horário, se existe reserva no horário todas as verificações com data, no método remover faz a verificação e validação para remover na linha 35. No método "salvar", a verificação de horário é realizada na linha 46 na Listagem 8.

Listagem 8 – ReservaServiceImpl.java

```
...
33  @Override
34 public void remover(Reserva reserva)throws NegocioException {
35         RestricaoHorario.permite(reserva.getDataFinal());
36         reservaRepository.delete(reserva);
37 }
38
39 @Override
40 @Transactional
41 public void salvar(Reserva reserva) {
42     Usuario usuario = usuarioService.porId(
43     reserva.getUsuario().getCodigo());
44     verificaUsuarioAtivo(usuario);
45     if (reserva.getCodigo() == null) {
46         RestricaoHorario.verificaHorarioReserva(
47         reserva ,reservaRepository.findAll()
48         );
49     }
50     verificaOpcaoOutro(reserva);
51     reserva.setUsuario(usuario);
52     reservaRepository.saveAndFlush(reserva);
53 }
54
55 private void verificaOpcaoOutro(Reserva reserva) {
56     if (reserva.getTipoevento().equals(TipoEvento.OUTROS) &&
57          reserva.getDescricao().isEmpty()) {
58  throw new NegocioException("Quando o tipo de evento e 'Outro' "
59                 + "\n Deverá comter a descrição do evento!!");
60     }
61 }
62
63 private void verificaUsuarioAtivo(Usuario usuario) {
64     if (usuario.getStatusUsuario() != StatusCadastro.ATIVO) {
65         throw new NegocioException("Usuario nao Ativo, procuro o
Administrador do sistema !!!");
66     }
```

```
67 }
...
```

Fonte: Autoria própria (2019).

Esta classe e responsável por fazer as validações com data. Para fazer os cálculos as datas são transformadas para o tipo *long*. Para fazer as restrições de 72 horas foi chegado a uma constante representado na linha 15 na Listagem 9. Para chegar a este valor foi pego duas datas, subtraído uma pela outra onde chegamos ao valor referente a um dia, representado na linha 13 na Listagem 9, sabendo que um dia tem 24 horas e que multiplicado por 3 chegamos a 72 horas. Então pegamos o valor e multiplicamos por 3 que chegamos ao valor da restrição em *long*. Na classe RestricaoHorario.java na Listagem 9, há o método responsável por validar se o usuário está de acordo com o termo de uso do sistema e verifica-o para as 72 horas para salvar e excluir onde entra com a data inicial e soma com a constante "restrição" da linha15, e verifica se é maior que a data atual mais 72 horas para frente.

Commented [m1]: Como chegou nesse valor?

Commented [DG2R1]: Ok

Listagem 9 – RestricaoHorario.java

```
...
8
9 public class RestricaoHorario implements Serializable {
10
11     private static final long serialVersionUID = 1L;
12
13     private static Long DIA = 86400000L;
14
15     private static Long RESTRICAO = 259200000L;
16
17     public static Long getRESTRICAO() {
18         return RESTRICAO;
19     }
20
21     public static Long getDia() {
22         return DIA;
23     }
24
25     public static boolean permite(Date dataInicial) {
26
27         Long hoje = new Date().getTime() + getRESTRICAO();
28
29         Long restricao = dataInicial.getTime();
30
31         if (hoje <= restricao) {
32             return true;
33         } else {
34          throw new NegocioException(
35                 "Não esta de acordo com o termo assinado!! "
36                  + "\n Esta fora do prazo de 72horas !!");
37         }
```

```
38
39      }
40
41      public static void verificaHorarioReserva (
42              Reserva reserva, List<Reserva> list) {
43
44          if (reserva.getCodigo() == null) {
45           Long iniRESERVA = reserva.getDataInicial().getTime();
46           Long fimRESERVA = reserva.getDataFinal().getTime();
47
48              permite(reserva.getDataInicial());
49
50              if (fimRESERVA <= iniRESERVA) {
51                    throw new NegocioException(
52                      "Data final e menor que data inicial !!");
53              }
54
55              List<Reserva> reservas = list;
56              for (Reserva reserva2 : reservas) {
57
58       Long iniRESERVA2 = reserva2.getDataInicial().getTime();
59       Long fimRESERVA2 = reserva2.getDataFinal().getTime();
60
61    if (iniRESERVA >= iniRESERVA2 && iniRESERVA <= fimRESERVA2) {
62          throw new NegocioException(
63                  "Ja Existe Reserva neste peiodo");
64              }
65    if (fimRESERVA >= iniRESERVA2 && fimRESERVA <= fimRESERVA2) {
66          throw new NegocioException(
67                  "Ja Existe Reserva neste peiodo");
68                }
69              }
70
71          }
72
73      }
74
75  }
76
```

Fonte: Autoria própria (2019).

O método verificaHorarioReserva na linha 41 da Listagem 9 verifica se a data é de início ou de fim, se está contida em alguma reserva existente.

A Listagem 10 mostra a classe responsável pelos envios dos *e-mails* tanto para o síndico quando para os moradores.

Listagem 10 – Email.java

```
...
11
12 @Component
13 public class Email implements Serializable {
14
```

```
15 private static final long serialVersionUID = 1L;
16
17 @Autowired
18 public JavaMailSender emailSender;
19
20 public void envia(Mensagem mensagem) {
21     SimpleMailMessage simpleMailMessage
22             = new SimpleMailMessage();
23
24     simpleMailMessage.setFrom(mensagem.getRemetente());
25     simpleMailMessage.setTo(mensagem.getDestinatarios()
26         .toArray(new String[mensagem.getDestinatarios()
27                 .size()]));
28     simpleMailMessage.setSubject(mensagem.getAssunto());
29     simpleMailMessage.setText(mensagem.getCorpo());
30
31     emailSender.send(simpleMailMessage);
32     }
33
34 }
35
...
```

Fonte: Autoria própria (2019).

A Listagem 11 mostra a classe EmailServiceImpl e seu métodos. Sendo que os métodos constam as anotações @Async e @EventListener onde as mesmas são responsáveis por escutar os eventos do Spring e executar em paralelo, não bloqueando o sistema. Toda vez que a implementação da interface ApplicationEventPublisher é chamada em algum lugar do sistema acaba disparando o método *publisher*.publishEvent, que envia um objeto para todos os métodos do sistema com a anotação @EventListener.

Listagem 11 – EmailServiceImpl.java

```
...
 22 @Component
 23 public class EmailServiceImpl implements EmailService {
 24
 25 @Autowired
 26 private UsuarioService usuarioService;
 27
 28 @Autowired
 29 private Email email;
 30
 31
 32 @Override
 33 @Async
 34 @EventListener
 35 public void enviarMoradia(UnidadeMoradia moradia) {
 36
 37 String corpo = "Ola " + moradia.getUsuario().getNome() +
 38     "\n Sua Unidade de Moradia : " + moradia.getUnidade()
```

```
 39      + "\n O Status :" + moradia.getStatusUnidadeMoradia()
 40              .getDescricao()
 + "\n O Sindico tem prazo de 72 horas para validar seu Cadastro,"
 42      + " você recebera um e-mail  assim que confirmando. ";
 43
 44      List<String> emails = new ArrayList<String>();
 45      emails.add(moradia.getUsuario().getEmail());
 46      Mensagem mensagem = new Mensagem(
 47          emails, moradia.getStatusUnidadeMoradia()
 48          + " Status do Cadastro da Moradia", corpo);
 49          email.envia(mensagem);
 50  }
 ...
 71  @Async
 72  @EventListener
 73  public void enviarSindico(Reserva reserva) {
 74
 SimpleDateFormat dt = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
 76      dt.setTimeZone(TimeZone.getTimeZone("America/Sao_Paulo"));
 77
 78      String dataInicial = dt.format(reserva.getDataInicial());
 79      String dataFinal = dt.format(reserva.getDataFinal());
 80
 81      String corpo;
 82if (reserva.getStatusReserva().equals(StatusReserva.EXCLUIDO)) {
 83      corpo = "Usuario : " + reserva.getUsuario().getNome()
 84          + "\n A Reserva para seguinte data : " + dataInicial
 85          + " a " + dataFinal + "\n Com Status :"
 86              + reserva.getStatusReserva().getDescricao();
 87      } else {
 88      corpo = "Usuario : " + reserva.getUsuario().getNome()
 89          + " , fez uma nova reserva."
 90          + "\n Reserva para seguinte data : "
 91          + dataInicial + " a " + dataFinal
 92          + "\n O Status :"+ reserva.getStatusReserva()
 93                  .getDescricao()
 94          + "\n Consulte o Sistema por o quanto antes.";
 95      }
 96      Usuario sindico = usuarioService.getSindico();
 97      List<String> emails = new ArrayList<String>();
 98      emails.add(sindico.getEmail());
 99      Mensagem mensagem = new Mensagem(
100         emails, reserva.getStatusReserva()
101         + " Status da Reserva : "
102         + reserva.getUsuario().getNome(), corpo);
103      email.envia(mensagem);
104  }
...
```

Fonte: Autoria própria (2019).

4 CONCLUSÃO

Este trabalho teve por objetivo desenvolver um sistema *web* para reservas de salão de festas em um condomínio, uma forma dinâmica de construir uma aplicação *web*, utilizando as tecnologias JSF, PrimesFaces e Spring. Contudo, ressalta-se que, assim como em qualquer projeto, foi indispensável o uso de fundamentos básicos da programação, como lógica de programação, análise de requisitos e construção de banco de dados.

No desenvolvimento deste trabalho o *back-end* foi construído implementando a biblioteca do ecossistema Spring como: Spring-Data, Spring-Security, Spring-Mail. Também foram utilizadas algumas especificações JEE como CDI utilizada para os escopos de vida dos *beans*. No *front-end* do projeto, responsável para inicializar a aplicação foi a biblioteca Adminfaces, que incorpora em um projeto Maven todas as bibliotecas para o funcionamento, além Adminfaces, do PrimeFaces na sua versão 7 e Mojarra 2.3.4 que são responsáveis pelo JSF.

Utilizando essas tecnologias foi possível aprimorar o conhecimento no desenvolvimento de aplicações Java *web* e conhecer novas tecnologias. Por fim, a realização deste trabalho também possibilitou aprimorar o raciocínio lógico no desenvolvimento das funcionalidades que estão no servidor, para realizar as operações e transições dos dados do banco de dados e as transições assíncronas.

Como trabalho futuro sugere-se melhorar as funcionalidades desenvolvidas. Além de acrescentar funcionalidades planejadas no início da construção deste projeto e de implementar controles para que em uma aplicação seja capaz de gerenciar um condomínio completo, desde as reservas até as receitas e seus gastos. E, ainda, possibilitar controle e acompanhamento mais detalhado pelos condôminos de consumo de gás e água, além de fiscalizar os gastos como um todo do condomínio.

REFERÊNCIAS

FARBER, J. C.; BOSCO, J. **Contribuição da contabilidade para a eficácia de gestão e controle de condomínios**. In: Congresso de Controladoria e Contabilidade da USP, 4. São Paulo. 2005.

GAMERS, William. **Visão Geral sobre PrimeFaces**. 2012. Disponível em: https://williamgamers.wordpress.com/2012/06/04/visao-geral-sobre-primefaces/. Acesso em: 14 nov. 2019.

JOINFACES. **JSF Spring Boot Starters**. Disponível em: http://joinfaces.org/. Acesso em: 14 nov. 2019.

MENDES, Maria C. D; MARTINS, Maria S. A. S. **Administração de condomínio na figura do síndico profissional**. Convibra 2015. Disponível em http://www.convibra.com.br/upload/paper/2015/31/2015_31_11951.pdf. Acesso em: 8 nov. 2019.

SOARES, Luiz. JSF – PARTE 1. **Revista Programar**. 2013. Disponível em: https://www.revista-programar.info/artigos/jsf-parte-1/. Acesso em: 14 nov. 2019.