UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAMON OLIVEIRA

Um aplicativo online para reservas de sala: o case do CINTED/UFRGS

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2018

RESUMO

A alocação de salas em instituições de ensino pode ser um tarefa bastante complexa, mesmo em setores menores com poucos usuários e salas disponíveis. Sem uma forma de restringir e controlar os horários dessas alocações, sobreposições de reservas podem acontecer e os intervalos definidos para cada usuário ficam mal gerenciados. Este trabalho tem como objetivo centralizar as reservas de salas do CINTED/UFRGS, tornando o gerenciamento dos horários mais transparente e robusto. Para tanto, descreve o desenvolvimento de um sistema Web para a gestão das salas do CINTED. O sistema foi avaliado, validado e colocado em uso.

Palavras-chave: Yii2 Framework. Web. Gerenciamento de alocação de salas.

An online app for room scheduling: The CINTED/UFRGS case

ABSTRACT

The allocation of classrooms in educational institutions can be a very complex task, even in smaller sectors with fewer users and available rooms. Without having a way of restricting and controlling the times of these allocations, reservations can be overlapped and the defined intervals for each user can be poorly managed. This work focuses on centralizing the scheduling of rooms in the CINTED/UFRGS, making the management of the schedule more transparent and robust. For this purpose, it describes the development of a Web system for managing the rooms in CINTED. The system was evaluated, validated and put to use.

Keywords: Yii2 Framework. Web Application; Room allocation management.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript And XML
CMS	Content Management System
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JS	JavaScript
JSON	JavaScript Object Notation
LAMP	Linux Apache MySQL PHP
MVC	Model-View-Controller
PHP	PHP: Hypertext Preprocessor
SQL	Structured Query Language
SUS	System Usability Scale

SUMÁRIO

1 INTRODUÇÃO

O Centro Interdisciplinar de Novas Tecnologias na Educação (CINTED), setor da Universidade Federal do Rio Grande do Sul (UFRGS), possui salas que podem ser reservadas para diversos tipos de aulas ou eventos. Existe a necessidade de gerenciar essas reservas de maneira clara e confiável - através de validações e controles de horários.

Múltiplos usuários efetuam as reservas e, atualmente, os agendamentos são feitos em planilhas que não possuem nenhuma forma de controle sobre os horários das salas, a não ser o cuidado dos próprios usuários na hora de marcar um horário.

Tendo em vista a necessidade de manter as informações organizadas, a ideia de criar um sistema para gerenciar esses agendamentos surgiu como uma forma de facilitar o trabalho e garantir que menos conflitos de horários venham a ocorrer. Sendo assim, o principal objetivo deste trabalho é criar a base de um novo sistema de reserva de salas para o CINTED, baseado em requisitos do próprio setor, para dar início ao que será um sistema mais completo e com mais funcionalidades no futuro. Uma implementação existente, o IAlocador[1], foi usada como inspiração para a criação desta aplicação.

O trabalho foi separado em 7 capítulos. O próximo capítulo aborda questões técnicas do desenvolvimento, mostrando as tecnologias escolhidas para a implementação e as ferramentas utilizadas no sistema. Na sequência, o terceiro capítulo traz o projeto do sistema, expondo a organização dos requisitos levantados e explicando as estruturas criadas para registrar os dados. O quarto capítulo descreve detalhes de implementação, explicando a arquitetura utilizada e os principais métodos criados. No quinto capítulo é apresentado um guia de utilização, que contém todas as telas do sistema, explicando suas funções e onde se encaixam no fluxo de uso. O sexto capítulo aborda os testes realizados, explicando a metodologia, além de listar e interpretar os dados obtidos nesses testes. Finalmente, no sétimo e último capítulo, são avaliados os objetivos iniciais do trabalho, analisando a contribuição e levantando sugestões para possíveis trabalhos futuros.

[1] https://www.inf.ufrgs.br/salas/

2 ESTADO DA ARTE, CONCEITOS E TECNOLOGIAS

Este capítulo discorre sobre as tecnologias e abordagens atuais para a implementação de sistemas Web, e como as ferramentas escolhidas foram utilizadas na criação do sistema de reserva de salas.

O capítulo contém três seções: ferramentas de desenvolvimento, *front-end* e *back-end* (as últimas duas sendo referentes às camadas da implementação do sistema), e cada seção possui subseções que explicam cada ferramenta ou tecnologia individualmente. A seção de ferramentas de desenvolvimento trata dos softwares/plataformas utilizados no planejamento, desenvolvimento e estruturação da aplicação. A segunda seção trata dos conceitos básicos relacionados à implementação da camada de *front-end*, que é a camada responsável pela construção da aplicação no lado do cliente, ou seja, códigos HTML, estilos CSS e a linguagem JavaScript. Por fim, a terceira seção apresenta os conceitos da camada de *back-end* da aplicação, que é a parte que roda no lado do servidor, sendo responsável pela lógica e tratamento dos dados, ou seja, códigos PHP e chamadas SQL, por exemplo.

2.1 Ferramentas de Desenvolvimento

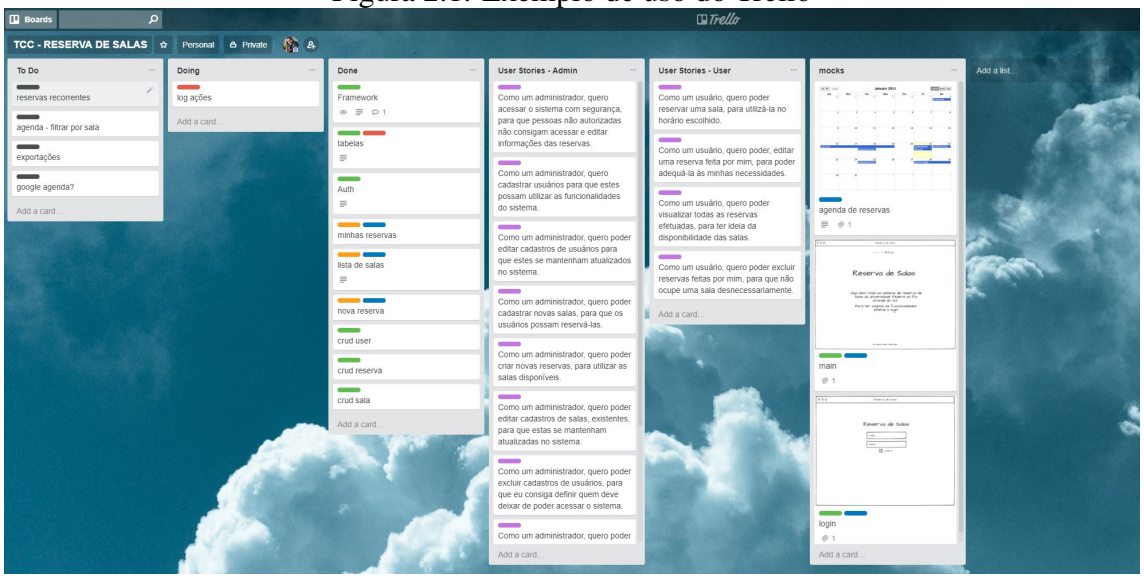
Nesta seção serão mostradas as principais ferramentas utilizadas no desenvolvimento do sistema de reservas. Cada subseção aborda uma ferramenta diferente, mostrando suas principais funções, alguns exemplos de uso (quando aplicável) e telas.

2.1.1 Trello

O Trello é uma ferramenta de colaboração para organização de projetos (TRELLO, 2017). A ferramenta é baseada em um quadro, onde cada cartão armazena uma tarefa que deve ser executada. A ideia é manter uma visão geral do projeto, facilitando a distribuição e o controle do andamento do desenvolvimento como um todo.

Esta implementação foi feita individualmente, não havendo um time propriamente dito, mas a ferramenta pode ser muito útil mesmo nesse cenário. As tarefas, ideias, *user stories* e quaisquer outro aspecto que necessita documentação foram todos registrados no Trello, o que auxiliou na visualização do projeto antes de iniciar o desenvolvimento.

Figura 2.1: Exemplo de uso do Trello



Fonte: Elaborada pelo autor

2.1.2 Visual Studio Code

O Visual Studio Code é um editor de código com suporte a diversas ferramentas de desenvolvimento, como integrações com GIT e *autocomplete* (MICROSOFT, 2018).

O uso desse editor auxilia no desenvolvimento em diversos aspectos, desde o entendimento de códigos mais complexos, devido ao suporte à sintaxe baseada em linguagens, até a integração com ferramentas de versionamento e ferramentas de *debugging*.

2.1.3 Github

O GitHub é uma plataforma de hospedagem de código, que facilita a contribuição em projetos privados ou *Open Source* à distância. Conta com controle de versão utilizando Git, permitindo a manutenção e extensão de aplicações com diversas ferramentas de controle (WIKIPEDIA, 2018b).

Apesar de ser mais extensamente utilizada, em termos de funcionalidades, em times com diversos desenvolvedores, também facilita o desenvolvimento para programadores em projetos individuais. Nesta implementação, além de manter o backup do código e facilitar a sincronização em múltiplas máquinas, permitiu também a manutenção do ambiente de testes, podendo atualizar o código que rodava na máquina virtual utilizada.

2.2 Yii2 Framework

O Yii2 é um framework PHP de alta performance baseado em componentes para desenvolvimento rápido e eficiente de aplicações Web. O nome Yii (pronunciado “Yee”, em inglês), significa “simples e evolucionário”, em chinês, ou serve como acrônimo de Yes It Is (YII, 2018c).

É um framework genérico, que pode ser usado todos os tipos de aplicações baseadas em Web. Por causa de sua arquitetura baseada em componentes e suporte a cache, é especialmente adequado para sistemas de larga escala, como portais, fóruns, sistemas de gerenciamento de conteúdos (CMS), e-commerce, sistemas Web RESTful etc (YII, 2018c).

Comparado a outros frameworks:

- Assim como diversos outros frameworks PHP, ele é baseado em uma arquitetura MVC (*Model-View-Controller*), o que auxilia na organização de lógica/código.
- O Yii segue a filosofia de que o código deve ser escrito de maneira simples e elegante. Sendo assim, em Yii, nada é modelado de forma exagerada, simplesmente para seguir algum padrão de design.
- É um framework *fullstack*, i.e., que envolve todas as camadas de desenvolvimento para a Web e que conta com diversas funcionalidades já testadas e muito utilizadas no mundo do desenvolvimento Web: construtores de *queries*, *Active Record* (para bancos de dados relacionais e NoSQL), suporte a RESTful e suporte a cache de vários níveis.

2.3 Front-End

Nesta seção são descritas as tecnologias e ferramentas de *front-end* utilizadas.

2.3.1 HTML/HTML5

HTML é a abreviação de HyperText Markup Language. É a linguagem usada para criar páginas Web. “HyperText” refere-se aos hiperlinks que uma página HTML contém, e que a conectam com outras páginas (MOZILLA, 2018c). “Markup Language” refere-se à maneira que as *tags* são utilizadas para definir o layout e os elementos da página.

HTML foi definida como padrão para criação de aplicações Web e é nativamente interpretada por todos navegadores atuais. No momento, encontra-se na quinta versão e suporta os mais recentes formatos de multimídia, incluindo vídeos, áudios e imagens.

Quando utilizada junto com ferramentas para criação de estilos, como a Castaded Style Sheets (CSS), e com ferramentas de gerenciamento de comportamento de páginas, como o JavaScript, torna-se uma poderosa ferramenta para construção de páginas Web.

2.3.2 CSS

CSS, abreviação para *Cascaded Style Sheets*, descreve como o elementos HTML devem ser mostrados na tela, papel ou em outras mídias. Pode ser usado para definir estilos de textos, formatos e tamanhos de tabelas, e diversos outros aspectos de uma página Web (MOZILLA, 2018b).

O CSS facilita o trabalho do desenvolvedor, tanto para unificar os estilos de todas páginas da aplicação em apenas um lugar, quanto para modificar a aparência geral do sistema.

Nesta implementação, o *framework open-source Bootstrap*[1] foi utilizado.

2.3.3 JavaScript

JavaScript é uma linguagem leve, interpretada e baseada em objetos, com funções de primeira classe, (MOZILLA, 2018d) conhecida como a linguagem de *scripting* (ou de programação) que permite a implementação de funcionalidades complexas em páginas Web. Acima do HTML e do CSS, faz com que as páginas sejam capazes de se comportar de forma mais dinâmica, podendo mostrar conteúdos dinamicamente, interagir com ações do usuário, mostrar gráficos e animações 2D/3D, etc.

Atualmente é a linguagem mais utilizada para definir a interação dos elementos com o usuário. Apesar de ser usada primariamente no lado do cliente, JavaScript tem crescido bastante no lado do servidor, através do framework Node.js[2]. A linguagem é extremamente versátil, permitindo que o desenvolvedor utilize diversos paradigmas, como programação orientada a objetos, programação imperativa e até programação funcional.

[1] http://getbootstrap.com/docs/3.3/
[2] https://nodejs.org/en/

2.3.4 JQuery

JQuery[3] é uma biblioteca que simplifica a interação do JavaScript com o HTML. É rápida, leve e possui muitas funcionalidades que tornam a manipulação de dados, tratamento de eventos, animações e chamadas AJAX mais simples, utilizando uma API compatível com a maioria dos navegadores modernos (FOUNDATION, 2018). Hoje em dia está altamente difundida, e grande parte dos programadores criam os códigos JavaScript através da JQuery ao invés do JavaScript puro.

Na implementação desta aplicação, a JQuery é responsável por quase toda interação dinâmica e chamadas AJAX, tornando o uso do sistema mais simples e intuitivo, diminuindo redirecionamentos e carregamentos de páginas.

2.3.5 Bootstrap

O Bootstrap é framework HTML, CSS e JavaScript, que facilita a criação de projetos responsivos ou para a *mobile* Web (BOLTCLOCK, 2011)[4]. É o framework Web mais popular, sendo usado como base para diversos tipos de aplicações Web de diversos tipos.

Por ser responsivo, bastante completo e usado extensivamente, possui uma variedade muito grande de temas e customizações que podem ser facilmente encontrados e aplicados, deixando o sistema mais organizado e consistente.

2.3.6 AJAX/JSON

AJAX é uma abreviação de *Asynchronous JavaScript and XML*. Em suma, é uma forma de de realizar chamadas assíncronas ao servidor da aplicação, através do objeto XMLHttpRequest (MOZILLA, 2018a).

O uso de chamadas assíncronas permite a criação de páginas que carregam seus dados de forma dinâmica, permitindo atualizar os elementos de forma independente sem requisitar um recarregamento da página inteira.

[3] https://jquery.com/
[4] http://stackoverflow.com/tags/twitter-bootstrap/info

2.3.7 Gii

Gii é uma extensão para o Yii2 Framework que facilita a geração de modelos, controladores, CRUDs, formulários e outras partes de um sistema. A criação desses componentes é feita baseada no banco de dados, e ocorre automaticamente, tornando mais simples a criação de funcionalidades básicas que costumam ser sempre muito parecidas (VEGIBIT, 2018).

Neste sistema, foram utilizados:

- Gerador de modelos: gera uma classe ActiveRecord para a tabela do banco de dados especificada.
- Gerador de CRUD : gera um controlador e as visualizações necessárias para implementar operações CRUD (Create Read Update Delete) referentes ao modelo (classe *ActiveRecord*) especificado.

2.4 Back-End

Nesta seção são descritas as tecnologias utilizadas no *back-end* da aplicação.

2.4.1 MySQL

MySQL é um sistema de gerenciamento de banco de dados baseado em SQL (*Structured Query Language*). Atualmente, é um dos sistemas de gerenciamento de banco de dados mais usado no mundo (WIKIPEDIA, 2018c).

O MySQL roda em praticamente todas as plataformas, incluindo Linux, UNIX e Windows. Apesar de poder ser usado em uma vasta variedade de aplicações, tem seu uso mais comum nas aplicações baseadas em Web. Junto com o Apache (servidor), Linux (sistema operacional) e PHP (linguagem orientada à objetos), forma um dos mais difundidos *stacks* de desenvolvimento Web *open source*, conhecido como LAMP.

2.4.2 ActiveRecord

Active Record é um padrão de arquitetura utilizado em software para guardar objetos de dados em bancos de dados relacionais (WIKIPEDIA, 2018a). A interface de um objeto que segue este padrão inclui funções de atualização (*Update*), inserção (*Insert*) e remoção (*Delete*), além de outras propriedades que correspondem às colunas da tabela do banco de dados.

Active Record é usado como uma forma de acessar os dados em um banco de dados. Uma tabela é referenciada através de uma classe, fazendo com que uma instância dessa classe seja uma linha da tabela. Todas ações feitas em um objeto refletem no banco de dados. Então, ao criar e salvar um objeto da classe, uma nova linha da tabela referenciada por essa classe é criada, e quando um objeto é modificado, a mudança é salva na linha da tabela.

Em Yii2, o ActiveRecord é uma classe base que implementa esse padrão, e ao invés de escrever comandos em SQL puro, o desenvolvedor acessa atributos do Active Record e chama métodos do Active Record para acessar e manipular dados salvos no banco de dados (YII, 2018a).

O Gii utiliza essa classe base para gerar novas classes, referentes às tabelas do banco de dados.

Uma classe *ActiveRecord* extende a classe base e implementa o método *tableName*:

```php
<?php

class Customer extends \yii\db\ActiveRecord
{
    public static function tableName()
    {
        return 'customer';
    }
}
```

2.4.3 Migrations

O Yii2 faz o uso de Migrations (migrações) para controlar alterações feitas no banco de dados. As migrações funcionam como uma espécie de versionamento de código, mas para mudanças do banco de dados.

No desenvolvimento e evolução de um sistema baseado em banco de dados, a estrutura do banco muda junto com o código da aplicação. Uma nova tabela pode se tornar necessária, ou uma mudança em algum índice para melhorar a performance de uma busca, por exemplo (YII, 2018b). As migrações são mudanças na estrutura do banco de dados descritas em forma de código, podendo ser versionadas juntamente com o código da aplicação.

2.4.4 MVC

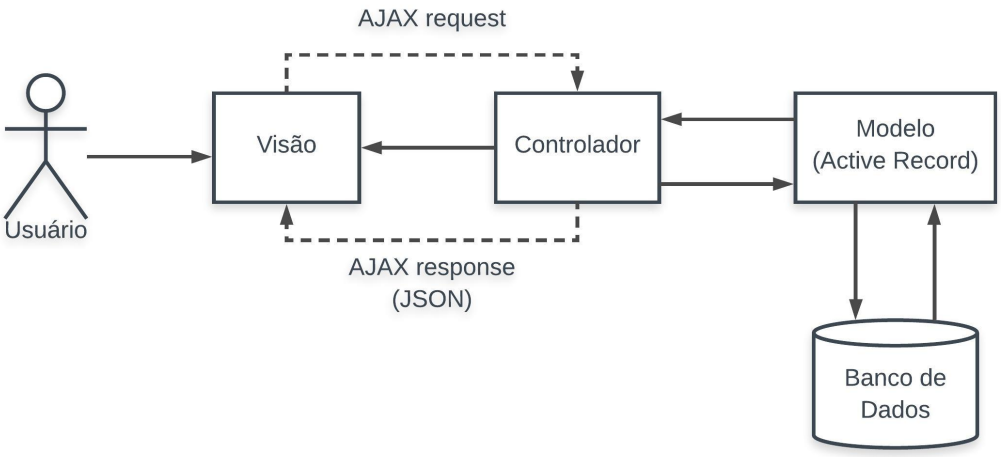
Model-View-Controller (em português, Modelo-Visão-Controlador) é um padrão de arquitetura de software que separa a representação e a interação (por parte do usuário) com os dados. Esse tipo de padrão geralmente é usado para a criação de interfaces de usuário, se tornou muito popular para o desenvolvimento de aplicações Web, além de outras plataformas (desktop, mobile, etc). Muitos frameworks de diversas linguagens já são desenvolvidos nesse padrão (REENSKAUG; COPLIEN, 2009).

As implementações variam entre linguagens e frameworks, mas a ideia geral do padrão é mantida. Geralmente, a divisão se dá da seguinte maneira:

- Modelo: é o componente central. Nele estão expressas as as regras de negócios e o comportamento da aplicação em termos de domínio de problema, independente da interação com o usuário, tratando diretamente os dados e lógicas do sistema;
- Visão: são todas as formas de interação com o usuário e visualização dos dados. As visões podem ser alteradas independentemente das regras de negócio, e o mesmo conjunto de dados pode ser mostrado de diversas maneiras diferentes;
- Controlador: é o intermediário da interação entre Modelo e Visão. Nele, as entradas são tratadas e convertidas em elementos e comandos para a visualização (DEACON, 1995).

O padrão MVC, como qualquer outro padrão de arquitetura, tem pontos fortes e fracos. A seguir lista-se alguns:

Figura 2.2: Padrão MVC



Fonte: Elaborada pelo autor

Vantagens:

- Desenvolvimento paralelo: como as lógicas são separadas e independentes, vários desenvolvedores podem trabalhar nos diferentes módulos de forma paralela;
- Coesão: o padrão mantém regras do mesmo domínio lógico no mesmo controlador, assim como agrupa as visões desse mesmo domínio;
- Baixo acoplamento: por definição, o padrão MVC possui baixo acoplamento entre models, visões e controladores;
- Facilidade de modificação/manutenção: por causa da separação das responsabilidades de cada parte do código, se torna fácil a manutenção e modificação/extensão.

Desvantagens:

- Complexidade de navegação do código: como é dividido em diversas camadas, e mais camadas tendem a ser introduzidas ao longo do desenvolvimento, a navegação pode se tornar complexa;
- Consistência: ao gerar uma nova funcionalidade, a lógica é dividida em três partes, podendo tornar difícil a manutenção da consistência na lógica e funcionalidade desejada;
- Curva de aprendizado: para programar em MVC, o conhecimento de várias tecnologias se torna necessário.

2.4.5 Composer

Composer é um gerenciador de dependências para o PHP que permite que as bibliotecas utilizadas no projeto sejam declaradas de forma centralizada, e gerencia atualizações e instalações das mesmas (COMPOSER, 2018).

Aplicações em PHP tendem a depender em diversos módulos e bibliotecas, e isso pode se tornar um trabalho complicado e desorganizado, visto que as dependências podem ocorrer em vários níveis (a aplicação depender de uma biblioteca, que depende de outra biblioteca, por exemplo). O Composer funciona por projeto, instalando as dependências em uma pasta junto ao código geral da aplicação. No Yii2, a pasta *vendor* é usada para esse propósito.

3 PROJETO DO SISTEMA

Neste capítulo são mostrados os detalhes do projeto do sistema. Primeiro, as funcionalidades serão descritas, no formato de *user stories*. Na sequência, as entidades do banco de dados MySQL serão apresentadas.

3.1 Funcionalidades

Esta seção lista as funcionalidades que foram levantadas para o sistema de reservas de salas. Essas funcionalidades estão apresentadas em *user stories*. Uma *user story* é uma descrição curta e simples de uma funcionalidade, da perspectiva de uma pessoa que deseja tal funcionalidade, geralmente um usuário ou cliente (COHN, 2004). Geralmente, elas são escritas no seguinte padrão:

Como um <tipo de usuário>, quero <algum objetivo> para que <alguma razão>.

Nesta implementação, tem-se dois tipos de usuário: *administrador* e *usuário*. Nas próximas seções são descritas as *user stories* para cada um dos tipos.

3.1.1 User Stories: Administrador

Nesta seção são apresentadas as histórias de usuário para o tipo administrador. Abaixo seguem as histórias de usuário:

- Como um administrador, quero acessar o sistema com segurança, para que pessoas não autorizadas não consigam acessar e editar informações das reservas.
- Como um administrador, quero cadastrar usuários para que estes possam utilizar as funcionalidades do sistema.
- Como um administrador, quero poder editar cadastros de usuários para que estes se mantenham atualizados no sistema.
- Como um administrador, quero poder cadastrar novas salas, para que os usuários possam reservá-las.
- Como um administrador, quero poder criar novas reservas, para utilizar as salas

disponíveis.

- Como um administrador, quero poder editar cadastros de salas, existentes, para que estas se mantenham atualizadas no sistema.
- Como um administrador, quero poder excluir cadastros de usuários, para que eu consiga definir quem deve deixar de poder acessar o sistema.
- Como um administrador, quero poder excluir cadastros de salas, para que eu consiga definir quais salas não podem mais ser reservadas.
- Como um administrador, quero poder excluir reservas, para ter controle sobre a situação geral das reservas.
- Como um administrador, quero poder acessar um histórico de exclusões das reservas, para poder auditar essas informações.

3.1.2 User Stories: Usuário

Nesta seção são apresentadas as histórias de usuário para o tipo usuário. Abaixo seguem as histórias de usuário:

- Como um usuário, quero poder reservar uma sala, para utilizá-la no horário escolhido.
- Como um usuário, quero poder editar uma reserva feita por mim, para poder adequá-la às minhas necessidades.
- Como um usuário, quero poder visualizar todas as reservas efetuadas, para ter ideia da disponibilidade das salas.
- Como um usuário, quero poder visualizar todas as reservas efetuadas por mim, para ter controle sobre elas de forma mais simples.
- Como um usuário, quero poder excluir reservas feitas por mim, para que não ocupe uma sala desnecessariamente.
- Como um usuário, quero poder filtrar as salas disponíveis, para poder reservá-la com mais facilidade.
- Como um usuário, quero poder editar minhas informações, para poder manter meu cadastro correto.

3.2 Modelagem do Banco de Dados

Nesta seção serão apresentadas as entidades que fazem parte do sistema. Em cada seção será detalhada uma das entidades do banco de dados, apresentando seus atributos e funcionalidades.

3.2.1 Entidade User

Esta entidade representa um usuário do sistema. Os campos desta entidade estão detalhados abaixo:

- **id**: identificador único gerado automaticamente pelo MySQL.
- **username**: campo do tipo STRING que armazena o nome do usuário.
- **auth_key**: token único de autenticação de acesso.
- **password_hash**: código *hash* criado pelo framework para encriptação da senha.
- **password_reset_token**: token gerado pelo framework para a recuperação da senha de acesso.
- **email**: endereço de email do usuário.
- **type_user**: campo do tipo STRING que determina o tipo de usuário.
- **status**: campo do tipo inteiro que representa o *status* do usuário (ativo ou inativo).
- **created_at**: campo do tipo inteiro que representa o *timestamp* da criação do registro.
- **updated_at**: campo do tipo inteiro que representa o *timestamp* da última modificação do registro.

3.2.2 Entidade Reserva

Esta entidade representa uma reserva. Os campos desta entidade estão detalhados abaixo:

- **id**: identificador único gerado automaticamente pelo MySQL.
- **sala_id**: chave estrangeira que referencia o identificador único da sala na entidade Sala.

- **user_id**: chave estrangeira que referencia o identificador único do usuário na entidade User.
- **titulo**: valor do tipo STRING que armazena o título descritivo da reserva.
- **descricao**: valor do tipo STRING que armazena a descrição e detalhes sobre a reserva.
- **dia**: campo do tipo DATE que armazena o dia (no formato Y-m-d) da reserva.
- **datahora**: campo do tipo DATETIME que armazena data e hora (no formato Y-m-d H:i:s) do início da reserva.
- **datafim**: campo do tipo DATETIME que armazena data e hora (no formato Y-m-d H:i:s) do fim da reserva.
- **duracao**: campo do tipo INTEGER que armazena o tempo em minutos da reserva.
- **created_at**: campo do tipo inteiro que representa o *timestamp* da criação do registro.
- **updated_at**: campo do tipo inteiro que representa o *timestamp* da última modificação do registro.

3.2.3 Entidade Sala

Esta entidade representa uma sala. Os campos desta entidade estão detalhados abaixo:

- **id**: identificador único gerado automaticamente pelo MySQL.
- **nome**: campo do tipo STRING que armazena o nome da sala.
- **tipo**: campo do tipo ENUM que armazena o tipo de sala.
- **hora_abertura**: campo do tipo TIME que armazena o horário (no formato Y-m-d) de abertura da sala.
- **hora_fechamento**: campo do tipo TIME que armazena o horário (no formato Y-m-d) de fechamento da sala.
- **created_at**: campo do tipo inteiro que representa o *timestamp* da criação do registro.
- **updated_at**: campo do tipo inteiro que representa o *timestamp* da última modificação do registro.

3.2.4 Entidade Log

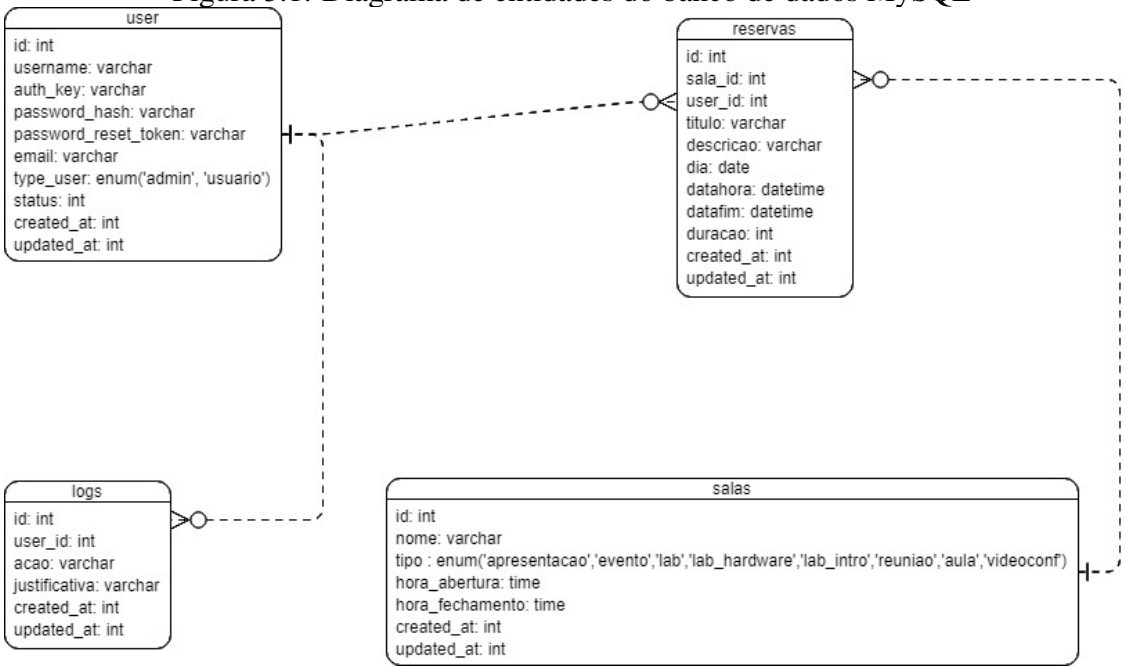
Esta entidade representa um log de ações. Os campos desta entidade estão detalhados abaixo:

- **id**: identificador único gerado automaticamente pelo MySQL.
- **acao**: valor do tipo STRING que armazena a ação sendo registrada.
- **descricao**: valor do tipo STRING que armazena o a justificativa da ação registrada.
- **user_id**: chave estrangeira que referencia o identificador único do usuário na entidade User.
- **created_at**: campo do tipo inteiro que representa o *timestamp* da criação do registro.
- **updated_at**: campo do tipo inteiro que representa o *timestamp* da última modificação do registro.

3.2.5 Diagrama de Entidades

Nesta seção é mostrado o diagrama de classes UML, para melhor visualização das entidades e seus relacionamentos. Cada classe representa uma tabela do banco de dados, e dentro delas estão detalhados seus campos (com os respectivos tipos).

Figura 3.1: Diagrama de entidades do banco de dados MySQL



Fonte: Elaborada pelo autor

4 IMPLEMENTAÇÃO DO PROJETO

Neste capítulo serão comentadas algumas das implementações principais utilizadas na camada de *Back-End* da aplicação, explicando os fluxos de tratamento de dados e redirecionamentos e algumas das funcionalidades centrais.

4.1 Arquitetura da aplicação

O modelo de arquitetura utilizada no sistema de reserva de salas é do tipo cliente-servidor. O conceito básico por trás desse modelo é a existência de pares de processos que se comunicam por mensagem transportadas através da rede. Cada par de processos em comunicação possui um processo rotulado como Cliente, e outro rotulado como Servidor. Se o usuário acessa uma aplicação pela Web, por exemplo, o navegador do qual ele acessa o sistema seria o processo Cliente, e o servidor no qual a aplicação está rodando seria o processo Servidor (KUROSE; ROSS, 2017).

Para esta aplicação, o processo considerado Cliente será o navegador do usuário, que troca mensagens com o processo Servidor sendo executado no servidor Web (utilizando o ambiente de testes como exemplo, a máquina virtual hospedada no sistema Google Cloud Compute).

Quando uma página é carregada no lado do Cliente, o navegador solicita a mesma ao servidor, acionando o controlador da aplicação, no processo Servidor, que devolve o código requisitado, juntamente com os dados, para ser exibido pelo navegador do Cliente.

Os dados trazidos pelo controlador são gerados através de chamadas ao banco de dados, feitas por intermédio dos modelo, utilizando o ActiveRecord.

No caso de requisições assíncronas (chamadas AJAX) - utilizadas para a fazer a filtragem da lista de salas na criação de uma nova reserva, por exemplo - a chamada vinda da visão no navegador do Cliente é recebida pelo controlador (no lado Servidor), que busca os dados através do modelo, e devolve para a visão, no formato JSON, e os dados são tratados através do JavaScript pelo navegador do Cliente, sem a necessidade de recarregar a visão por inteiro.

A Figura 2.2, além de mostrar o padrão MVC de arquitetura, dá uma visão geral sobre o fluxo descrito anteriormente.

4.2 CRUDs

Para a criação dos cadastros de cada modelo (Usuário, Sala e Reserva), os controladores possuem métodos que implementam as funcionalidades de CRUD (Create Read Update Delete - ou Criar Ler Atualizar Remover). Através da ferramenta de criação de código Gii, fornecida pelo Yii2, a criação do código base para essas operações é feita de forma automática e mais genérica, permitindo que os métodos sejam apenas editados conforme a necessidade de cada modelo.

Por padrão, o Yii2 trata as requisições de GET e POST no mesmo método, e as URLs de acesso são baseadas no nome do métodos. Sendo assim, os métodos tem o nome prefixado por *action*, e o nome da ação (que também se refere ao endereço de acesso). Por exemplo, o método de criação se chamaria *actionCreate*. A seguir serão mostradas capturas de tela que mostram um código de exemplo de cada uma dessas ações.

4.2.1 Create

O método Create é responsável por criar uma nova instância do modelo e salvá-la no banco de dados.

```
1     /**
2      * Creates a new Salas model.
3      * If creation is successful, the browser will be redirected to the
      'view' page.
4      * @return mixed
5      */
6     public function actionCreate()
7     {
8         $model = new Salas();
9
10
11        if ($model->load(Yii::$app->request->post()) && $model->save())
      {
12            Yii::$app->session->setFlash('success', 'Sala salva com
    sucesso!');
13            return $this->redirect(['view', 'id' => \$model->id]);
14        }
15
16        return $this->render('create', [
```

```
17              'model' => $model,
18          ]);
19      }
```

Como mostra o código acima, uma nova instância do modelo é atribuída para a variável *$model*, e é testado se existe uma requisição POST e se o modelo pode ser salvo. Caso o teste falhe, significa que é uma requisição GET, e a aplicação redireciona o usuário para a tela de cadastro. Caso o teste passe, significa que é uma requisição POST, e os dados fornecidos pelo usuário são salvos no banco de dados. Após o salvamento, o usuário é redirecionado para a tela de visualização do cadastro recém finalizado.

4.2.2 Read

O método View é responsável por ler os dados de uma instância do modelo existente no banco de dados.

```
1       /**
2        * Displays a single Salas model.
3        * @param integer $id
4        * @return mixed
5        * @throws NotFoundHttpException if the model cannot be found
6        */
7       public function actionView($id)
8       {
9           return $this->render('view', [
10              'model' => $this->findModel($id),
11          ]);
12      }
```

Como mostra o código acima, o método de leitura é apenas uma requisição do tipo GET, já que não precisa coletar dados do usuário. Por isso, o parâmetro *$id*, passado para o método, é usado para encontrar o registro no banco de dados, e as informações do mesmo são carregadas na tela de visualização. Para encontrar o registro, é usado o método *findModel($id)*, mostrado a seguir.

```
1       /**
2        * Finds the Salas model based on its primary key value.
3        * If the model is not found, a 404 HTTP exception will be thrown.
4        * @param integer $id
5        * @return Salas the loaded model
```

```
6       * @throws NotFoundHttpException if the model cannot be found
7       */
8      protected function findModel($id)
9      {
10         if (($model = Salas::findOne($id)) !== null) {
11             return $model;
12         }
13
14         throw new NotFoundHttpException('The requested page does not
       exist.');
15     }
```

4.2.3 Update

O método Update é responsável por editar uma instância existente do modelo e salvá-la no banco de dados. Se comporta de forma parecida com o método Create.

```
1      /**
2       * Updates an existing Salas model.
3       * If update is successful, the browser will be redirected to the '
       view' page.
4       * @param integer $id
5       * @return mixed
6       * @throws NotFoundHttpException if the model cannot be found
7       */
8      public function actionUpdate($id)
9      {
10         $model = $this->findModel($id);
11
12         if ($model->load(Yii::$app->request->post()) && $model->save())
       {
13             Yii::$app->session->setFlash('success', 'Sala editada com
       sucesso!');
14             return $this->redirect(['view', 'id' => $model->id]);
15         }
16
17         return $this->render('update', [
18             'model' => $model,
19         ]);
20     }
```

Como mostra o código acima, a instância do modelo, buscada através do método *findModel($id)*, é atribuída para a variável *$model*, e é testado se existe uma requisição POST e se o modelo pode ser salvo. Caso o teste falhe, significa que é uma requisição GET, e a aplicação redireciona o usuário para a tela de edição, com os dados existentes. Caso o teste passe, significa que é uma requisição POST, e os dados fornecidos pelo usuário (modificados ou não) são salvos no banco de dados. Após o salvamento, o usuário é redirecionado para a tela de visualização do cadastro recém finalizado.

4.2.4 Delete

O método Delete é responsável por apagar um registro do modelo existente do banco de dados.

```
1     /**
2      * Deletes an existing Salas model.
3      * If deletion is successful, the browser will be redirected to the
      'index' page.
4      * @param integer $id
5      * @return mixed
6      * @throws NotFoundHttpException if the model cannot be found
7      */
8     public function actionDelete($id)
9     {
10         $this->findModel($id)->delete();
11         Yii::$app->session->setFlash('success', 'Sala removida com
    sucesso!');
12
13         return $this->redirect(['index']);
14     }
```

Como mostra o código acima, o método de remoção é apenas uma requisição do tipo GET, já que não precisa coletar dados do usuário. Por isso, o parâmetro *$id*, passado para o método, é usado para encontrar o registro no banco de dados, e as informações do mesmo são carregadas na tela de visualização. Para encontrar o registro, é usado o método *findModel($id)*. Logo após, é chamado o método *delete()*, que apaga o registro do banco de dados. Após a remoção, o usuário é redirecionado para a tela inicial do controlador (index).

4.3 Ambiente de Testes

Para possibilitar os testes, a aplicação foi configurada e instalada em uma instância de máquina virtual do Google Cloud Compute[1]. O serviço permite a criação de máquinas virtuais para diversos fins, com várias configurações diferentes para se adequar ao uso da aplicação.

O sistema foi mantido em modo de desenvolvimento, mesmo no servidor, para facilitar as correções que poderiam se tornar necessárias enquanto os testes fossem feitos. No servidor, a *stack* LAMP (Linux Apache MySQL PHP) foi utilizada.

[1] https://cloud.google.com/compute/

5 INTERFACES DE NAVEGAÇÃO

Este capítulo tem como objetivo mostrar as telas que compõem o sistema de reserva de salas.

5.1 Sistema Web

Nesta seção são apresentadas imagens das telas do sistema e comentadas suas funções no fluxo de utilização do sistema.

5.1.1 Tela de início

Ao acessar o sistema, o usuário visualiza a tela inicial do sistema, que contém o nome da aplicação, uma frase de boas vindas e um link para acessar o sistema (tanto no corpo da tela quanto na barra superior de navegação), como mostra a Figura 5.1.

Figura 5.1: Tela Inicial

Fonte: Elaborada pelo autor

5.1.2 Tela de login

A tela de login (como mostra a Figura 5.2) é exibida quando o usuário seleciona a opção "Login"na barra de navegação do sistema, ou na interface da página inicial (quando não há nenhum login efetuado). Para ter acesso ao sistema, o usuário deve possuir credenciais previamente cadastradas por um administrador. Após o login ser efetuado, o usuário é redirecionado novamente para a tela inicial, mas agora tem acesso às funcionalidades do sistema (baseado no seu perfil de usuário).

A tela principal passa a mostrar três botões (como mostra a Figura 5.3), que levam às funcionalidades principais da aplicação (Lista de Salas, Agenda de Reservas e Lista de Reservas), e a barra superior de navegação passa a mostrar todos links de acesso (no caso do usuário do tipo Administrador: Usuários | Salas | Agenda | Lista de Reservas | Minhas Reservas | Trocar Senha | Logout; no caso do usuário do tipo Usuário: Salas | Agenda | Lista de Reservas | Minhas Reservas | Trocar Senha | Logout).

Figura 5.2: Tela de Login

Fonte: Elaborada pelo autor

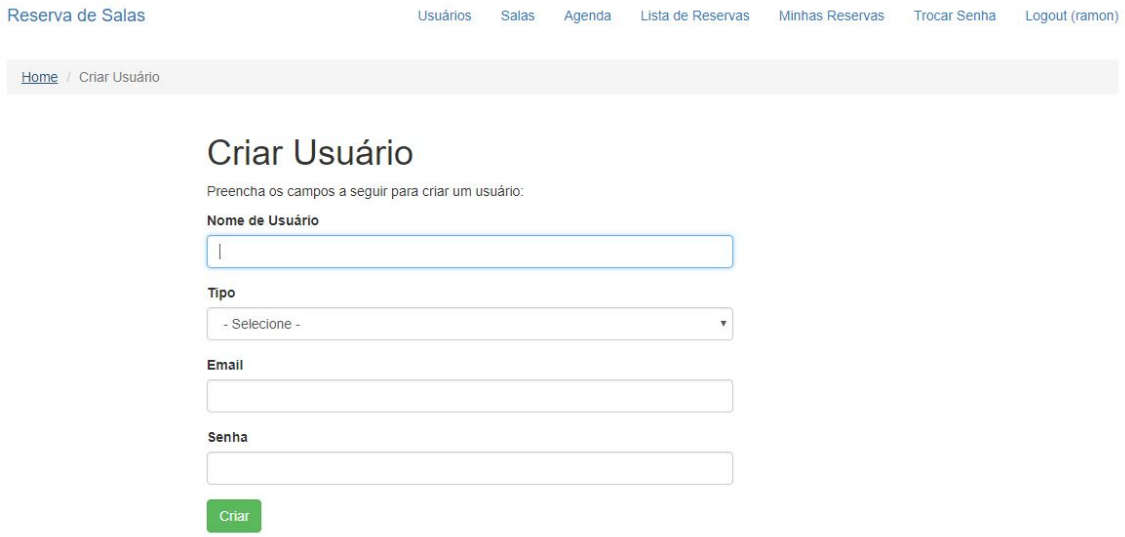
Figura 5.3: Tela Inicial pós Login

Fonte: Elaborada pelo autor

5.1.3 Tela de cadastro de usuário

A tela de cadastro de usuário (mostrada na Figura 5.4) é visualizada apenas por administradores, e é acessível pela listagem de usuários. Nela, é possível cadastrar um novo usuário, definindo seu nome, tipo (administrador ou usuário), e-mail e senha temporária.

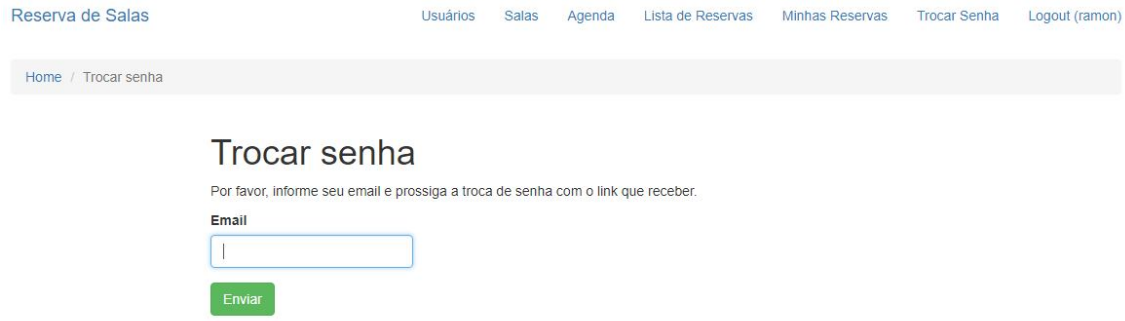
Figura 5.4: Tela de cadastro de usuário



Fonte: Elaborada pelo autor

Após o cadastro ser efetuado, é dada a opção ao usuário de trocar sua senha (Figura 5.5) (através da barra de navegação superior).

Figura 5.5: Tela de troca de senha

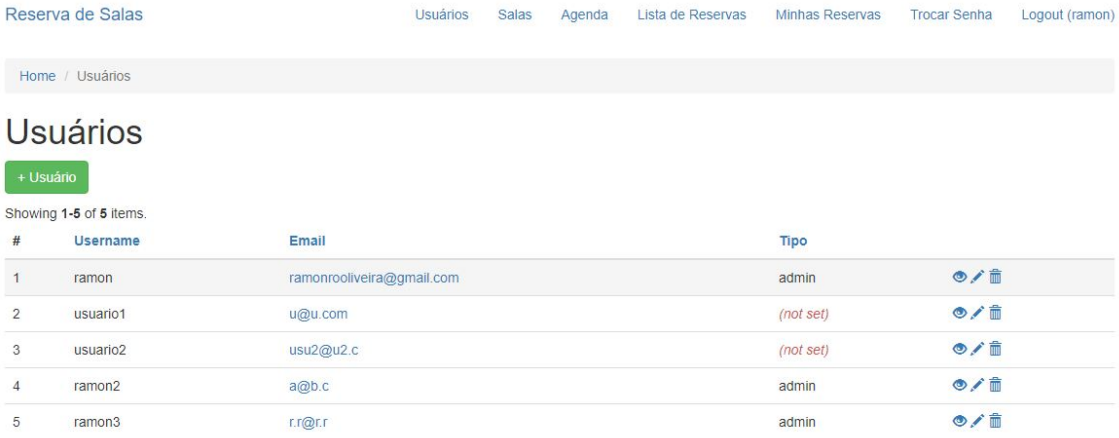


Fonte: Elaborada pelo autor

5.1.4 Tela de listagem de usuário

Nesta tela, acessível pelo cabeçalho do sistema, o administrador pode visualizar a listagem de usuários cadastrados no sistema, tendo acesso à informações dos usuários como nome, email e tipo de usuário. A partir dela, pode-se acessar outras funcionalidades ligadas aos usuários, como a criação, a edição, a visualização e a remoção.

Figura 5.6: Tela de listagem de usuário



Fonte: Elaborada pelo autor

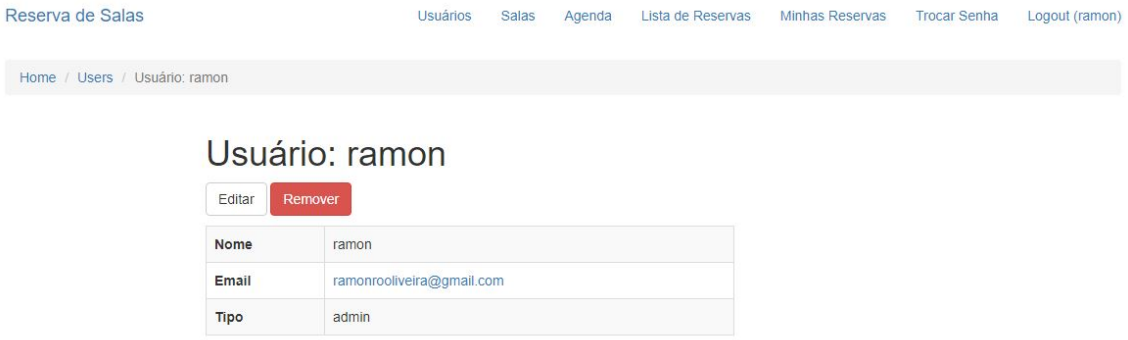
5.1.5 Tela de visualização de usuário

Nesta tela, acessível através da listagem de usuários, ou logo após a criação de um usuário novo, tem-se o resumo das informações do usuário sendo visualizado. A partir dela, pode-se acessar a tela de edição do cadastro do usuário, ou removê-lo.

5.1.6 Tela de edição de usuário

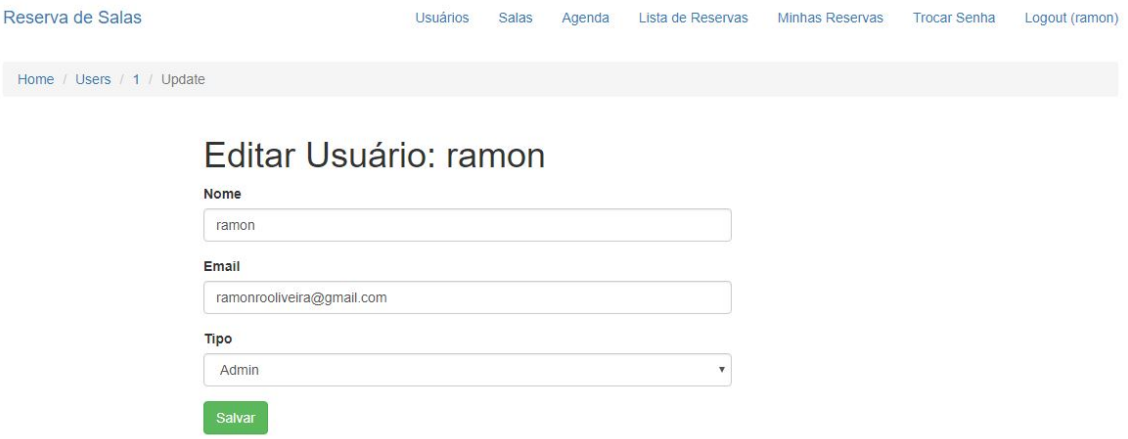
Nesta tela, acessível através da listagem de usuários ou da tela de visualização de usuário (apenas para administradores), é possível editar as informações de nome, email e tipo de um usuário cadastrado. Ao clicar no botão editar, o administrador é redirecionado para a tela de edição (mostrada na Figura 5.8), onde pode atualizar as informações e salvar as alterações, clicando no botão Salvar.

Figura 5.7: Tela de visualização de usuário



Fonte: Elaborada pelo autor

Figura 5.8: Tela de edição de usuário

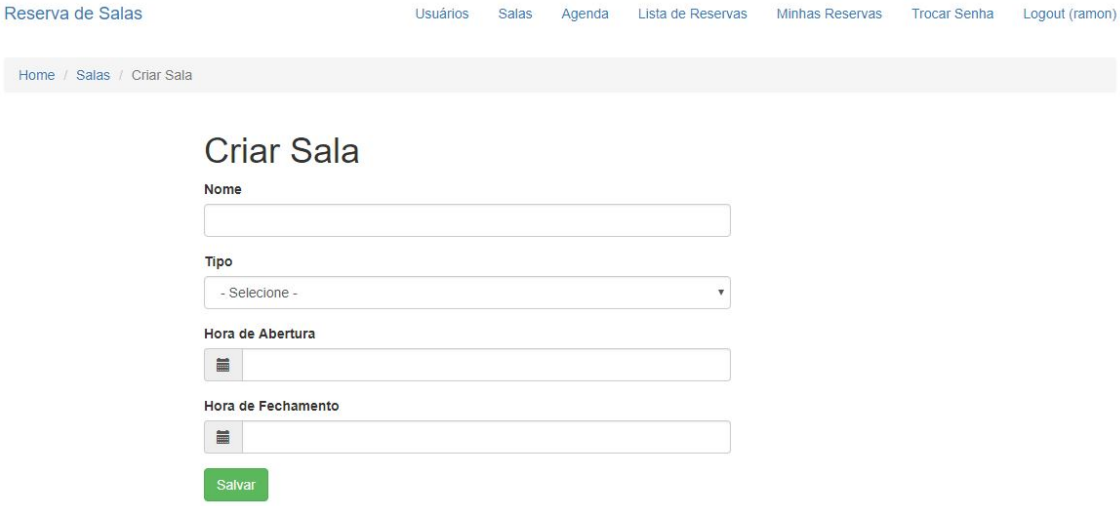


Fonte: Elaborada pelo autor

5.1.7 Tela de cadastro de salas

Nesta tela (mostrada na Figura 5.9), acessível através da tela de listagem de salas, apenas por administradores, é possível criar o cadastro de uma nova sala. As informações que podem ser adicionadas são: nome da sala, tipo (escolhido de uma lista de tipos), horário de abertura da sala e horário de fechamento da sala. Ao salvar o novo cadastro, o usuário é redirecionada à tela de visualização da sala criada.

Figura 5.9: Tela de cadastro de salas

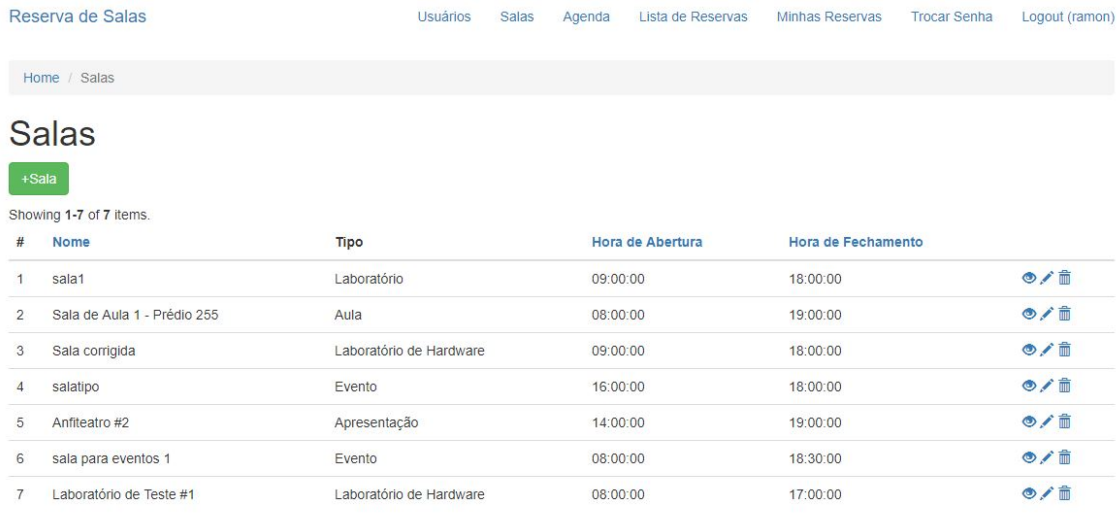


Fonte: Elaborada pelo autor

5.1.8 Tela de listagem de salas

Nesta tela (mostrada na Figura 5.10), acessível através da barra de navegação superior do sistema, o usuário pode ver a listagem das salas cadastradas no sistema, mostrando informações de cada sala como: nome, tipo de sala, horário de abertura e horário de fechamento. A partir desta tela, o usuário pode visualizar uma sala, editar uma sala ou remover uma sala (no caso da edição e da remoção, apenas se o usuário for administrador - os botões de edição e remoção são omitidos caso o usuário logado seja do tipo Usuário).

Figura 5.10: Tela de listagem de salas



Fonte: Elaborada pelo autor

5.1.9 Tela de visualização de salas

Nesta tela (mostrada na Figura 5.11), acessível após salvar uma nova sala, ou através da listagem de salas, o usuário pode visualizar um resumo das informações da sala desejada. A partir dela, utilizando os botões Editar e Remover (omitido para usuário do tipo Usuário), o administrador pode, respectivamente, atualizar as informações da reserva, ou remover a reserva do banco de dados.

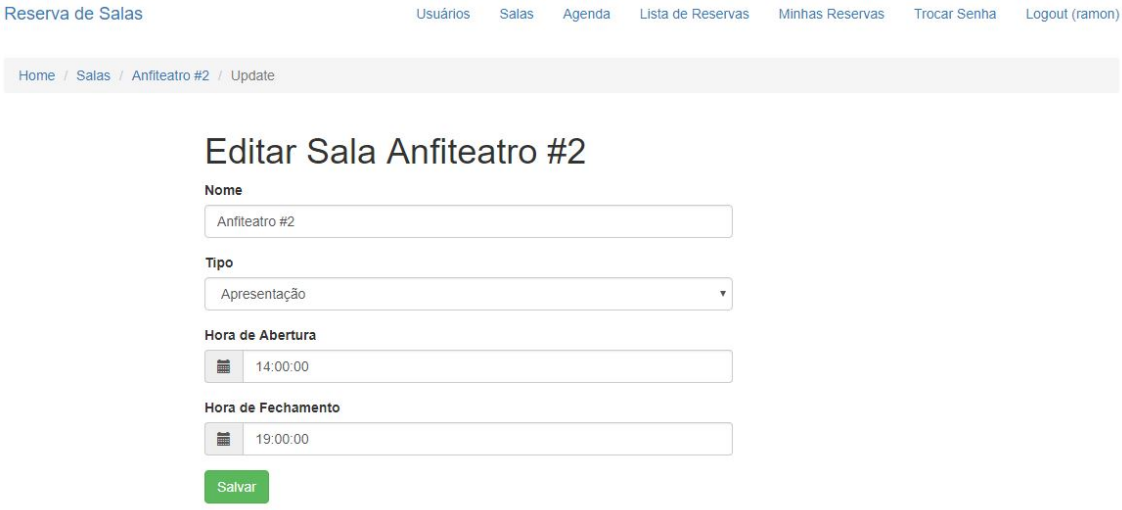
Figura 5.11: Tela de visualização de salas

Fonte: Elaborada pelo autor

5.1.10 Tela de edição de salas

Esta tela (mostrada na Figura 5.12), acessível através da listagem de salas ou através da tela de visualização de sala, permite que administrador modifique as informações de nome, tipo de sala, hora de abertura e hora de fechamento da sala cadastrada.

Figura 5.12: Tela de edição de salas



Fonte: Elaborada pelo autor

5.1.11 Tela de cadastro de reservas

Esta tela (mostrada na Figura 5.13), acessível pela lista de reservas, ou pela agenda de reservas (tanto clicando no botão +Reserva, ou clicando no dia que deseja-se criar a reserva), permite que o usuário inicie um cadastro de uma nova reserva.

Esta tela engloba também a pesquisa por salas, baseada nas informações de horário, que filtra as salas disponíveis usando as restrições informadas, com o intuito de evitar conflitos de utilização.

O usuário ainda pode escolher a opção de recorrência, que permite que a reserva seja feita em mais de um dia simultaneamente. Nesse caso, deve-se escolher os dias da semana e o número de semanas que se deseja a recorrência, (p. ex. Figura 5.15) e o salvamento é feito automaticamente.

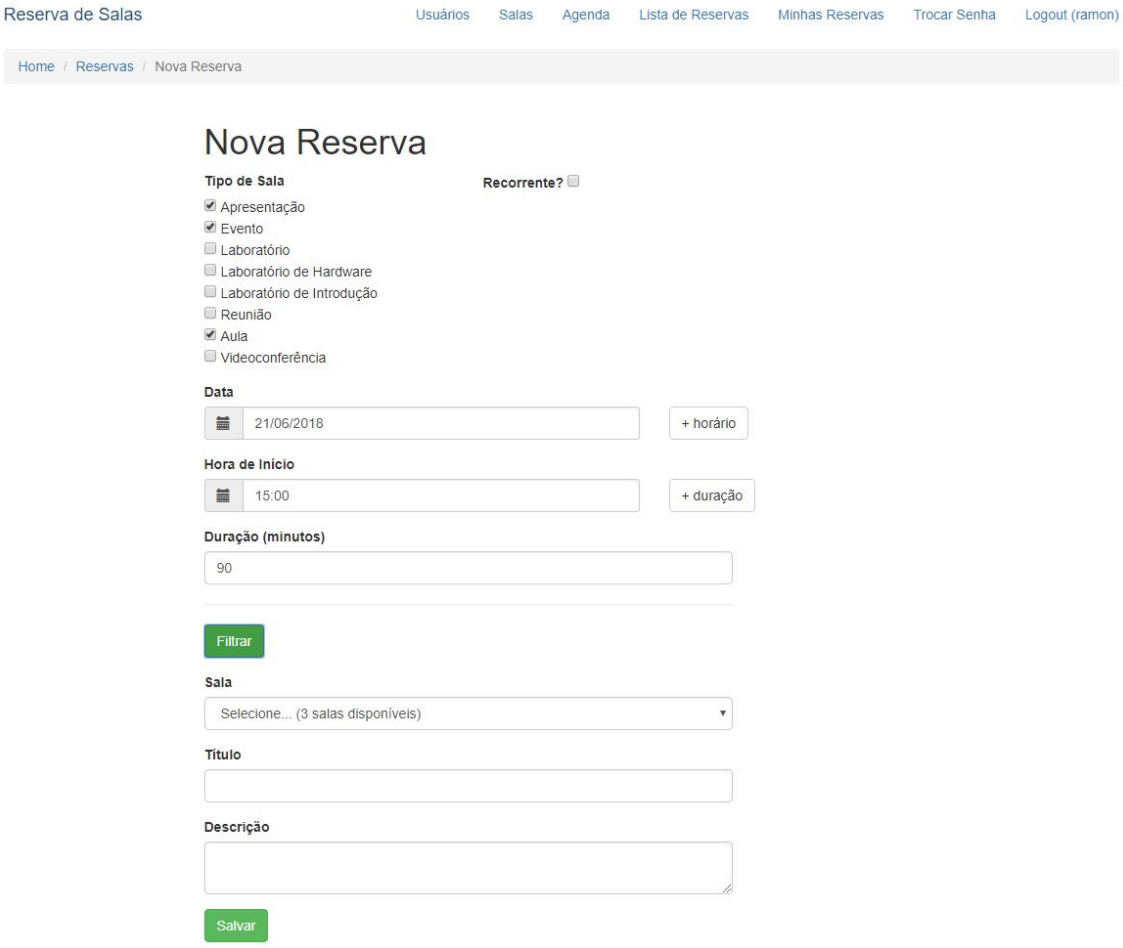
A filtragem das salas disponíveis se dá da seguinte maneira: ao carregar a tela de cadastro, o usuário tem a opção de escolher os tipo de sala (em uma lista de *checkboxes*) e a data (utilizando um componente de *DatePicker*, conforme a Figura 5.16) que deseja reservar uma sala.

Ao clicar no botão +horário, um campo de horário (com um *TimePicker*, mostrado na Figura 5.17) é mostrado, para a escolha do horário inicial da reserva. Após, clicando em +duração, um campo de entrada de texto é mostrado, onde espera-se uma duração (em minutos) para a reserva.

Figura 5.13: Tela de cadastro de reservas

Fonte: Elaborada pelo autor

Figura 5.14: Tela de cadastro de reservas - Filtragem



Fonte: Elaborada pelo autor

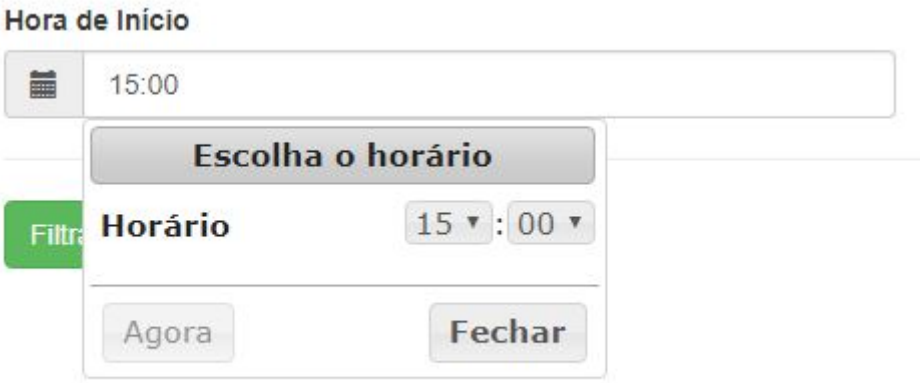
Figura 5.15: Tela de cadastro de reservas - Recorrência

Fonte: Elaborada pelo autor

Figura 5.16: Datepicker

Fonte: Elaborada pelo autor

Figura 5.17: Timepicker



Fonte: Elaborada pelo autor

Baseando-se nesse campos, é chamado um método de filtragem (mostrado nas figuras 5.18 e 5.19), via AJAX, que busca todas as salas que se encaixam nas restrições, excluindo as salas que já possuam reservas que colidem em algum dos requisitos. Por exemplo, se uma reserva já existe para o mesmo dia desejado, das 10h da manhã, com duração de 60 minutos, e o usuário quiser iniciar sua reserva às 10h30, a sala que possui esta reserva das 10h às 11h não aparecerá na listagem, evitando conflito de utilização.

A seguir é possível ver os detalhes do método de filtragem de salas:

Figura 5.18: Método que encontra colisões

```
// retorna todas colisões existentes no horário selecionado
$reservas_existentes = Reservas::find()->select(['sala_id'])
                    ->joinWith([
                        'sala' => function($query) use ($tipo_sala) {
                            $query->where(['in', 'tipo', $tipo_sala]);
                        },
                    ], false)
                    ->where(['dia' => $dia_sql])
                    ->andFilterWhere(['>', 'datafim', $datahora])
                    ->andFilterWhere(['<','datahora', $datahora_fim])
                    ->asArray()
                    ->all();
```

Fonte: Elaborada pelo autor

Figura 5.19: Método que encontra salas excluindo colisões

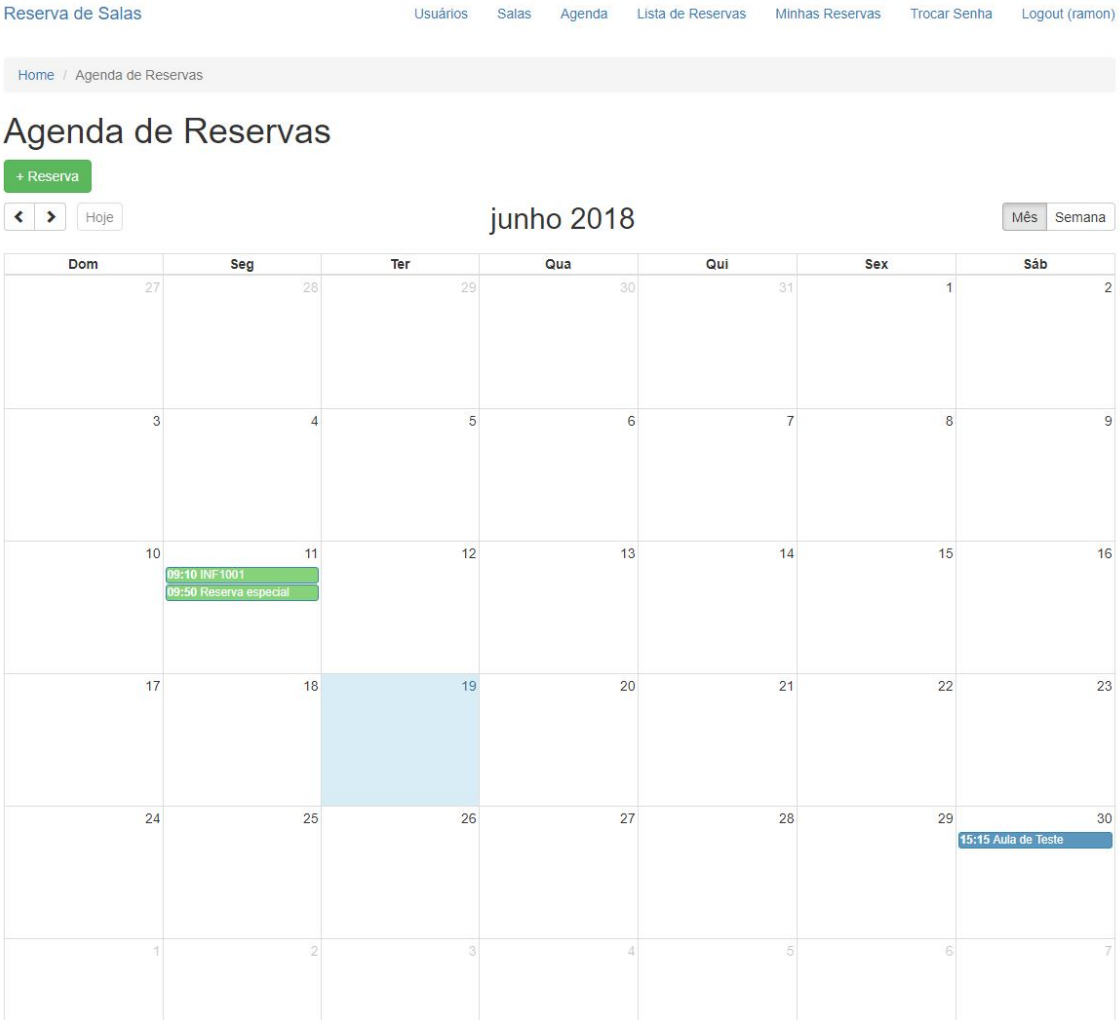
```
$salas = Salas::find()
            ->where(['NOT IN', 'id', $colisoes])
            ->andFilterWhere(['in', 'tipo', $tipo_sala])
            ->andFilterWhere(['<=', 'hora_abertura', $hora_inicio])
            ->andFilterWhere(['>=', 'hora_fechamento', $hora_inicio])
            ->andFilterWhere(['>=', 'hora_fechamento', $hora_fim])
            ->all();
```

Fonte: Elaborada pelo autor

5.1.12 Agenda de reservas

A tela da agenda de reservas (Figura 5.20) é acessível pela tela inicial e pela barra de navegação superior, e é o componente central para a visualização das reservas de forma geral. Baseada no componente Jquery FullCalendar (YII, 2018d), a agenda mostra todas as reservas efetuadas, por mês ou por semana, em forma de eventos, como em um calendário tradicional. Cada reserva aparece no quadrado referente ao dia para qual a reserva foi feita, com uma cor referente ao seu turno (azul para Manhã, amarelo para Tarde). A partir dessa agenda, o usuário pode visualizar uma reserva feita, clicando nela, o que abre a possibilidade de Editar ou Remover a mesma (caso seja um administrador ou caso a reserva tenha sido feita pelo usuário logado). Além disso, pode-se iniciar uma nova reserva, clicando no botão +Reserva, ou no dia em que se deseja reservar (neste caso, uma janela de confirmação mostra o dia que foi selecionado e, em caso afirmativo, o usuário é redirecionado para a tela de cadastro de reservas.

Figura 5.20: Agenda de reservas

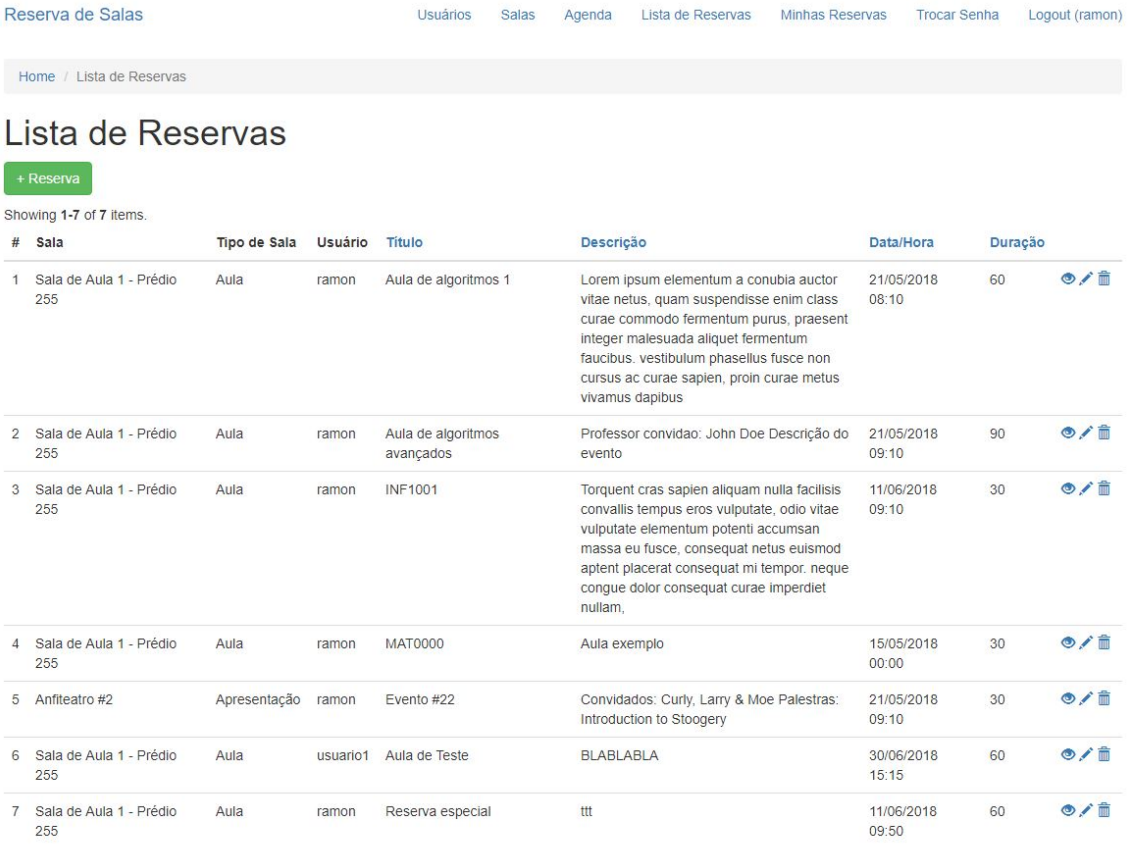


Fonte: Elaborada pelo autor

5.1.13 Tela de listagem de reservas

Nesta tela (mostrada na Figura 5.21), acessível através da tela inicial ou da barra de navegação superior, o usuário tem acesso à lista geral de reservas efetuadas, podendo ver informações importantes de todas as reservas feitas até o momento (nome, sala, responsável, horário, duração), além de poder iniciar uma nova reserva (clicando no botão +Reserva), visualizar uma reserva, editar ou remover uma reserva (sendo as duas últimas acessíveis apenas para administradores, ou caso a reserva em questão tenha sido criada pelo usuário logado).

Figura 5.21: Tela de listagem de reservas

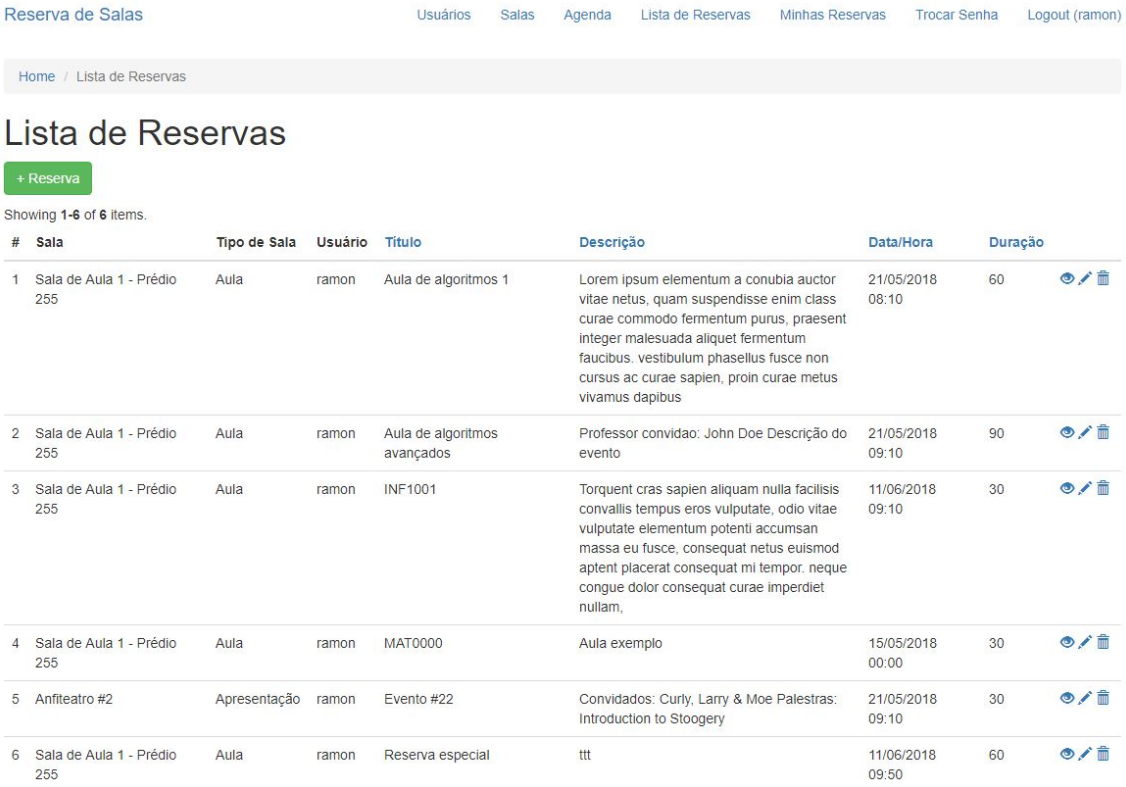


Fonte: Elaborada pelo autor

5.1.14 Tela de listagem - Minhas reservas

Esta tela (mostrada na Figura 5.22), acessível através da barra de navegação superior, permite ao usuário filtrar apenas reservas que ele mesmo criou. A estrutura da tabela e as informações são as mesmas da lista geral de reservas. Através dessa tela o usuário pode iniciar uma reserva (clicando em +Reserva), visualizar, editar ou remover (e pelo fato de as reservas aqui listadas serem todas de sua autoria, o usuário tem acesso à todas funcionalidades, mesmo não sendo administrador).

Figura 5.22: Tela de listagem - Minhas reservas



Fonte: Elaborada pelo autor

5.1.15 Tela de visualização de reservas

Essa tela (mostrada na Figura 5.23, acessível através da agenda de reservas (clicando em uma reserva existente, nesse caso, é aberta uma janela *modal* com as informações), e da lista de reserva, mostra as principais informações de uma reserva existente, como título da reserva, nome da sala, usuário responsável, descrição, data e hora de início e duração em minutos. A partir dela, o usuário pode editar ou remover a reserva (caso seja um administrador, ou seja uma reserva criada por si), clicando em Editar ou Remover, respectivamente. Caso não tenha acesso a essas funcionalidades, os botões são omitidos.

Figura 5.23: Tela de visualização de reservas

Fonte: Elaborada pelo autor

5.1.16 Tela de edição de reservas

Nesta tela (mostrada na Figura 5.24), acessível através da agenda de reservas (clicando em uma reserva existente, e depois clicando em Editar), ou através da lista de reservas, o usuário pode atualizar informações da reserva. Neste caso, apenas o usuário responsável, o título e a descrição podem ser atualizados, pois a mudança de horário poderia causar diversos problemas de conflito de horário, e a decisão tomada no fluxo da aplicação para manter os horários coerentes foi de que os horários não devem ser atualizados. Para modificar o horário de uma reserva, o usuário deve apagar a reserva já realizada e iniciar uma nova, assim, possíveis conflitos serão detectados e evitados.

Figura 5.24: Tela de edição de reservas

Fonte: Elaborada pelo autor

6 ESTUDO DE CASO: EXPERIMENTO ESPECÍFICO E AVALIAÇÃO

Após a implementação de todas as funcionalidades principais do sistema, foram realizados testes com usuários, com o objetivo de validar a usabilidade da aplicação. A seguir esses testes são detalhados.

6.1 Formato dos Testes

Os testes implementados focam na usabilidade e foram formulados usando como base a SUS (System Usability Scale - ou Escala de Usabilidade de Sistemas). A SUS permite efetuar testes de forma "rápida e suja"(do inglês, "*quick and dirty*"), mas confiável, para medir a usabilidade de um sistema (USABILITY.GOV, 2018).

Criada em 1986, a SUS permite a avaliação de diversos produtos e serviços, desde *hardwares* até aplicações Web.

Para gerar os testes baseados na SUS, é criado um questionário com 10 questões, cada uma com cinco opções de respostas, variando de "discordo plenamente" a "concordo plenamente".

A escala SUS tem como alguns de seus benefícios:

- É uma escala muito simples de usar com os usuários;
- Pode ser usada em populações pequenas com resultados confiáveis;
- É válida, podendo diferenciar sistemas usáveis e não usáveis (USABILITY.GOV, 2018);

O fator que torna a SUS confiável e adequada para a pequena população de usuários com a qual os testes foram realizados é sua popularidade e a quantidade de dados existentes para basear os resultados, tornando-os mais significativos (SAURO, 2011).

A SUS não possui uma forma única de analisar os resultados, e também não define o que seria um resultado médio aceitável. Mas o fato de ser usada de forma tão vasta e ter tantos dados de testes anteriores torna possível a normalização e a definição de um ponto de corte. Baseando-se em resultados de mais de 5000 usuários e cerca de 500 estudos sobre diversos tipos de aplicação, o consenso é de que o resultado médio seja 68 (SAURO, 2011). Sendo assim, resultados acima de 68 podem ser considerados acima da média, e abaixo de 68, abaixo da média.

Algumas considerações a se fazer sobre a SUS:

- Os resultados puros não são percentis; logo, um resultado de 70 na SUS não significa uma nota equivalente à 7 de 10. (Este item será melhor explicado na seção de interpretação dos resultados).
- A SUS não é um diagnóstico do sistema e não é feita para encontrar falhas na usabilidade; é uma maneira de classificar o sistema em termos de facilidade de uso e de aprendizado.

6.2 Interpretação dos Resultados

Nesta seção, os resultados dos testes feitos baseado na escala SUS serão comentados.

Os testes foram feitos com 8 usuários, sendo 2 deles funcionários do CINTED, 3 usuários que trabalham na área de TI, e 3 usuários que não trabalham na área de TI (e possuem pouco contato com sistemas similares ao desse trabalho).

O questionário foi criado com uma lista de tarefas principais a serem executadas. Após a realização dessas tarefas, o usuário responde as 10 questões da SUS, e pode fazer comentários gerais sobre a aplicação e seu uso.

A Tabela 6.1 sumariza os resultados do questionário SUS de forma geral, com a escala simplificada para 3 níveis (concordância, neutralidade e discordância), enquanto as figuras 6.1 e 6.2 mostram os resultados separados por tipo de pergunta.

Como os resultados mostram, a avaliação do sistema apresentou uma nota média na escala SUS altamente satifatória, com 78.75 pontos de média entre os 8 testes realizados. A questão com maior discrepância nas respostas foi a questão 4 (Eu acho que precisaria apoio de um técnico para poder usar esse sistema), na qual metade dos respondentes se posicionaram a favor da afirmação ou de forma neutra. Isso pode ser analisado do ponto de vista de que apenas dois dos respondentes trabalham no CINTED, então alguns dos comportamentos do sistema podem não ter ficado claros o suficiente para os outros respondentes, que não estão acostumados com o fluxo de trabalho.

Além disso, nenhum dos respondentes achou o sistema desnecessariamente complexo, e todos os respondentes acharam que as funcionalidades da aplicação ficaram bem integradas.

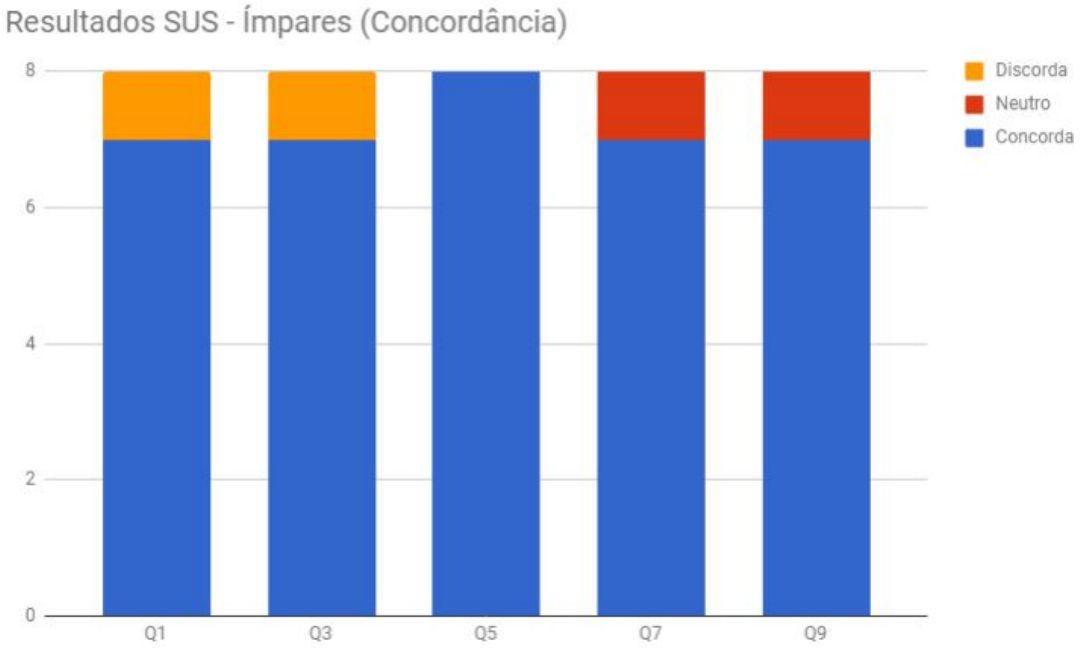
Analisando a curva de *ranking* de resultados da escala SUS, mostrada na figura 6.3, e usando como base os pontos levantados por Jeff Sauro (SAURO, 2011), conclui-se que o resultado médio ficaria por volta de um percentil 85, ou seja, a aplicação conseguiu

Tabela 6.1: Sumário das respostas do questionário SUS

Questão	Afirmação	Concorda	Neutro	Discorda
Q1	Eu acho que gostaria de usar o sistema frequentemente.	87.5%	0%	12.5%
Q2	Achei o sistema desnecessariamente complexo.	0%	0%	100%
Q3	Achei o sistema fácil de usar.	87.5%	0%	12.5%
Q4	Eu acho que precisaria apoio de um técnico para poder usar esse sistema.	25%	25%	50%
Q5	Achei que as várias funcionalidades do sistema estão bem integradas.	100%	0%	0%
Q6	Encontrei muitas inconsistências no sistema.	0%	0%	100%
Q7	Imagino que grande parte das pessoas aprenderia a usar esse sistema rapidamente.	87,5%	12,5%	0%
Q8	Achei o sistema bastante complicado.	0%	12.5%	87.5%
Q9	Me senti bastante confiante ao usar o sistema.	87.5%	12.5%	0%
Q10	Precisei aprender muitas coisas antes de usar o sistema.	12.5%	0%	87.5%

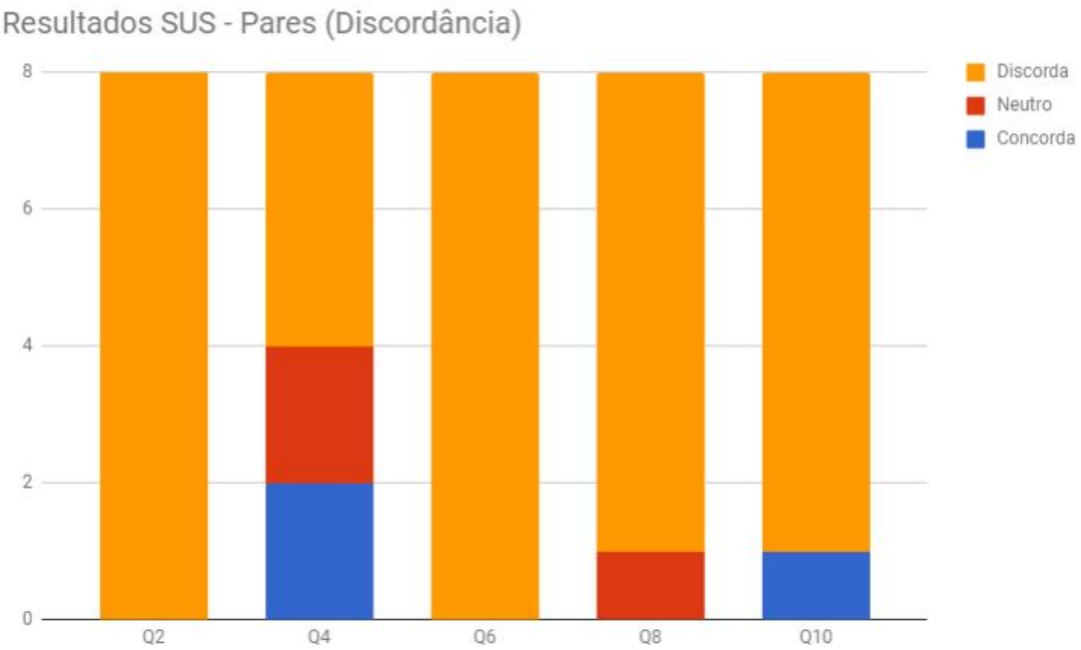
uma pontuação maior do que 85% das aplicações testadas, e usando um sistema de notas baseado em letras, sua nota seria perto de A.

Figura 6.1: Resultados SUS - Ímpares (Concordância



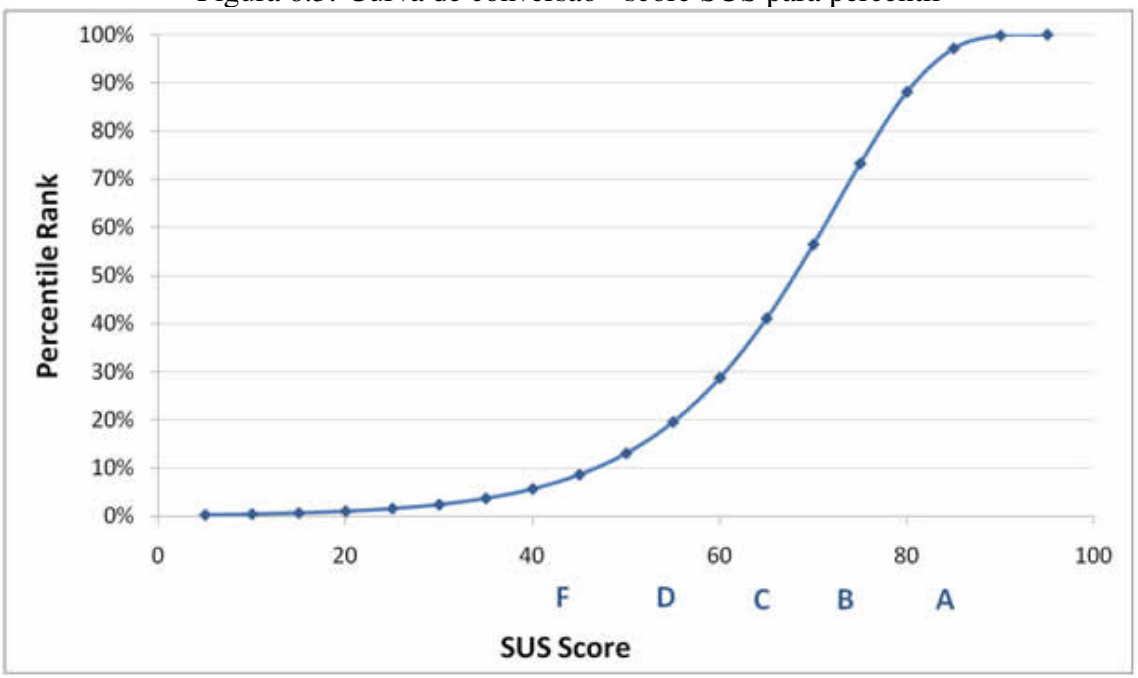
Fonte: Elaborada pelo autor

Figura 6.2: Resultados SUS - Pares (Discordância



Fonte: Elaborada pelo autor

Figura 6.3: Curva de conversão - score SUS para percentil



Fonte: Jeff Sauro - uxpamagazine.org/sustified

7 CONCLUSÃO E TRABALHOS FUTUROS

A realização do sistema de reserva de salas do CINTED surgiu de uma necessidade de manter as informações concisas, organizadas e sem conflitos, sendo simples de ser usada e adotada no fluxo atual dos processos internos do setor. Desde as decisões em termos tecnológicos, pensando em ferramentas que facilitem a integração do sistema no CINTED, e que possibilitem a expansão para outros setores da universidade, até às formas de funcionamento, tentando encaixar da melhor forma possível no trabalho que já é executado.

Os testes demonstraram que o sistema conseguiu atingir seu objetivo de ser simples e efetivo, mantendo o fluxo o mais direto possível mas ainda sim executando as tarefas que se espera. Além disso, as tecnologias escolhidas para o desenvolvimento se mostraram adequadas para atingir o objetivo principal do sistema e facilitam sua extensão em próximas iterações.

Este trabalho deixa em aberto a adição de algumas funcionalidades que ainda serão desenvolvidas e adaptadas, como, por exemplo, o controle para evitar a posterção indefinida (o caso de um usuário nunca conseguir a reserva ou conjunto de reservas que deseja). Em um futuro próximo, o sistema sofrerá alterações para que fique cada vez mais adequado às necessidades específicas do dia a dia do CINTED, e começará a ser utilizado para auxiliar na organização.

REFERÊNCIAS

BOLTCLOCK, S. O. **About: twitter-bootstrap**. 2011. <https://stackoverflow.com/tags/twitter-bootstrap/info>.

COHN, M. **User Stories Applied: For Agile Software Development**. [S.l.]: Addison Wesley, 2004.

COMPOSER. **Composer - Introduction**. 2018. <https://getcomposer.org/doc/>.

DEACON, J. Model-view-controller (mvc) architecture. **JOHN DEACON Computer Systems Development, Consulting Training**, aug 1995.

FOUNDATION jQuery. **What is jQuery?** 2018. <https://jquery.com>.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores: uma abordagem top-down**. [S.l.]: Pearson, 2017. ISBN 978-85-430-1443-2.

MICROSOFT. **Why did we build Visual Studio Code?** 2018. <https://code.visualstudio.com/docs/editor/whyvscode>.

MOZILLA. **AJAX: Getting Started**. 2018. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started>.

MOZILLA. **CSS**. 2018. <https://developer.mozilla.org/pt-BR/docs/Web/CSS>.

MOZILLA. **HTML: HyperText Markup Language**. 2018. <https://developer.mozilla.org/en-US/docs/Web/HTML>.

MOZILLA. **JavaScript**. 2018. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.

REENSKAUG, T.; COPLIEN, J. **The DCI Architecture: A New Vision of Object-Oriented Programming**. 2009. <https://www.artima.com/articles/dci_vision.html>.

SAURO, J. **SUStisfied? Little-Known System Usability Scale Facts.** 2011. <http://uxpamagazine.org/sustified>.

TRELLO. **Trello Help - What is Trello?** 2017. <https://www.yiiframework.com/doc/guide/2.0/en/intro-yii>.

USABILITY.GOV. **System Usability Scale (SUS)**. 2018. <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. [Online: Acessado 28 de Agosto de 2015].

VEGIBIT. **Super Easy CRUD With Gii And The Yii2 Framework**. 2018. <https://vegibit.com/super-easy-crud-with-gii-and-the-yii2-framework/>.

WIKIPEDIA. **Active record pattern**. 2018. <https://en.wikipedia.org/wiki/Active_record_pattern>.

WIKIPEDIA. **GitHub**. 2018. <https://pt.wikipedia.org/wiki/GitHub>.

WIKIPEDIA. **MySQL**. 2018. <https://pt.wikipedia.org/wiki/MySQL>.

YII. **The Definitive Guide to Yii 2.0 - Active Record**. 2018. <https://www.yiiframework.com/doc/guide/2.0/en/db-active-record>.

YII. **The Definitive Guide to Yii 2.0 - Database Migration**. 2018. <https://www.yiiframework.com/doc/guide/2.0/en/db-migrations>.

YII. **The Definitive Guide to Yii 2.0 - What is Yii**. 2018. <https://www.yiiframework.com/doc/guide/2.0/en/intro-yii>.

YII. **The Definitive Guide to Yii 2.0 - What is Yii**. 2018. <https://www.yiiframework.com/doc/guide/2.0/en/intro-yii>.