auditorium Documentation

Versão 0.0.1

Evandro Murilo, Valdeir Cesconeto

24 mar, 2018

Ajuda:

CAPÍTULO 1

Sobre

Auditorium é um sistema de agendamento de auditórios feito sob demanda para o CEULJI/ULBRA, como trabalho final da disciplina de Tópicos Especiais III (frameworks) ministrada pelo professor Edgar Kaiser.

CAPÍTULO 2

Models

2.1 Auditorium

`Auditorium` é o model que representa um auditório. Sua tabela correspondente no banco de dados se chama `auditoria`.

2.1.1 Atributos

name **`VARCHAR(20)`** Nome do auditório

capacity **`INTEGER`** Quantas pessoas o auditório comporta.

accessible **`BOOLEAN`** Se o auditório preenche os requisitos de acessibilidade.

location **`VARCHAR(100)`** Localização do auditório.

obs **`TEXT`** Observações adicionais sobre o auditório.

2.1.2 Métodos

statusOn(**`Carbon $date`** **)** Retorna um objeto `Status` representando o status do auditório na data `$date`.

Ver: *Status*.

2.1.3 Views

index View onde são exibidos todos os auditórios e seus status para determinado dia.

Recebe como parâmetro GET opcional uma `data`; caso nenhuma seja fornecida, utiliza a data atual. Também recebe os parâmetros `previous` e `next`, que implementam a funcionalidade das setas.

2.2 Request

`Request` é o model que representa um pedido de agendamento de auditório. Sua tabela correspondente no banco de dados se chama `requests`.

2.2.1 Atributos

auditorium_id `INTEGER` ID do auditório.

user_id `INTEGER` ID do usuário que requisitou o agendamento.

period `TINYINT` Período do agendamento:

0. manhã
1. tarde
2. noite

date `DATE` Data do agendamento.

event `VARCHAR(100)` Nome que identifica o agendamento, preferencialmente o nome do evento.

description `TEXT` Descrição do pedido de agendamento.

status `TINYINT` Status do pedido:

0. pendente
1. rejeitado
2. aceito

2.2.2 Views

create View do formulário de criação de uma nova `Request`.

Recebe como parâmetros GET obrigatórios a `data` atual e o `id` do auditório a ser agendado, além de um parâmetro opcional `period` com um código de período.

index View onde as `Requests` são exibidas e editadas. Recebe como parâmetro GET opcional um `filter` que pode ter um dos seguintes valores:

all Todas as `Requests` são exibidas.

pendent São exibidas somente as `Requests` pendentes. É o valor padrão.

resolved São exibidas somente as `Requests` que não estão mais pendentes.

accepted São exibidas somente as `Requests` que foram aceitas.

rejected São exibidas somente as `Requests` que foram negadas.

show View que mostra os detalhes de uma `Request` em específico.

2.3 User

`User` é o model que representa um usuário. Sua tabela correspondente no banco de dados se chama `users`.

2.3.1 Atributos

name `VARCHAR` Nome do usuário.

email `VARCHAR` Email do usuário, utilizado para o login. Deve ser único.

password `VARCHAR` Senha do usuário, utilizada para o login.

description `TEXT` Descrição do usuário, idealmente seu cargo. Ex.: «Coordenador de Sistemas de Informação».

color `CHAR(7)` Código de cor do avatar do usuário em formato hexadecimal.

cel `VARCHAR(19)` Número de telefone do usuário.

2.3.2 Views

show Perfil do usuário.

index Lista de usuários.

edit Página de edição do usuário.

2.4 Call

`Call` é o model que representa uma chamada. Sua tabela correspondente no banco de dados se chama `calls`.

2.4.1 Atributos

title `VARCHAR(40)` Título da chamada.

user_to_user `TINYINT` Se é uma chamada de um para um.

2.4.2 Componentes

Call.vue Componente principal de uma chamada.

CallMember.vue Componente que representa um usuário na lista de usuários na chamada.

CallMessage.vue Componente que representa uma mensagem na lista de mensagens na chamada.

NewCall.vue Componente para criação de chamadas.

2.5 Message

`Message` é o model que representa uma mensagem em uma `Call`. Sua tabela correspondente no banco de dados se chama `messages`.

2.5.1 Atributos

call_id `INTEGER` ID da chamada a qual essa mensagem pertence.

user_id `INTEGER` ID do usuário que enviou a mensagem.

body `TEXT` Corpo da mensagem.

CAPÍTULO 3

Helpers

3.1 Status

A classe `Status` representa o status de um auditório em uma data específica.

3.1.1 Atributos

morning, **afternoon** e **night** representam o status do auditório no determinado período:

0. pendente
1. disponível
2. indisponível

morning `Int` Representa o status do auditório no período da manhã.

afternoon `Int` Representa o status do auditório no período da tarde.

night `Int` Representa o status do auditório no período da noite.

date `Carbon` A data do `Status` em questão.

requests `Collection` Uma `Collection` contendo todas as `Requests` existentes para o auditório em questão.

morning_requests `Collection` Uma `Collection` contendo todas as `Requests` existentes para o auditório em questão no período da manhã.

afternoon_requests `Collection` Uma `Collection` contendo todas as `Requests` existentes para o auditório em questão no período da tarde.

night_requests `Collection` Uma `Collection` contendo todas as `Requests` existentes para o auditório em questão no período da noite.

3.1.2 Exemplo

```
// retorna o código de status do auditório no período da noite de hoje
$auditorium->statusOn(Carbon::now())->night
```

CAPÍTULO 4

Events

4.1 RequestStatusChanged

`RequestStatusChanged` é um evento disparado quando o status de uma `Request` é alterado.

4.2 CallCreated

`CallCreated` é um evento disparado quando uma nova chamada é criada.

4.3 NotificationRead

`NotificationRead` é um evento disparado quando uma notificação é lida. É enviado via *broadcasting* no canal `App.User.{userId}`.

4.3.1 Atributos

user `User` O usuário que leu a notificação.

4.4 RequestCreated

`RequestCreated` é um evento disparado quando um novo pedido de auditório é feito.

4.5 UserRegistered

`UserRegistered` é um evento disparado quando um usuário é registrado.

4.6 MessageCreated

`MessageCreated` é um evento disparado quando uma nova mensagem é enviada.

CAPÍTULO 5

Listeners

5.1 SendStatusChangeNotification

`SendStatusChangeNotification` é um listener que ouve eventos do tipo `RequestStatusChanged` e envia uma notificação do tipo `RequestResolved` para o usuário que fez a `Request`.

5.2 AssignNewUserToChat

`AssignNewUserToChat` é um listener que ouve eventos do tipo `UserRegistered` e adiciona uma permissão para que ele participe do chat universal.

5.3 SendNewMessageNotification

`SendNewMessageNotification` é um listener que ouve eventos do tipo `MessageCreated` e envia uma notificação do tipo `NewMessage` para todos os envolvidos.

5.4 SendNewRequestNotification

`SendNewRequestNotification` é um listener que ouve eventos do tipo `RequestCreated` e envia uma notificação do tipo `NewRequest` para os usuários que tem permissão de secretário.

5.5 SetNewUserRole

`SetNewUserRole` é um listener que ouve eventos do tipo `UserRegistered` e adiciona permissões de coordenador para o usuário recém registrado.

5.6 SetCallMemberPermissions

`SetCallMemberPermissions` é um listener que ouve eventos do tipo `CallCreated` e adiciona permissões para que todos os usuários envolvidos possam ver a chamada.

5.7 SendNewCallNotification

`SendNewCallNotification` é um listener que ouve eventos do tipo `CallCreated` e envia uma notificação do tipo `NewCall` para todos os envolvidos.

CAPÍTULO 6

Notifications

6.1 RequestResolved

`RequestResolved` é uma notificação enviada quando o status de uma `Request` é modificado. É armazenada no banco de dados e enviada via *broadcasting* no canal `App.User.{userId}`, para o usuário dono da `Request`.

6.2 NewCall

`NewCall` é uma notificação enviada quando uma nova chamada é criada. É armazenada no banco de dados e enviada via *broadcasting* no canal `App.User.{userId}`, para todos os usuários envolvidos na chamada.

6.3 NewMessage

`NewMessage` é uma notificação enviada quando uma nova mensagem é recebida. É armazenada no banco de dados e enviada via *broadcasting* no canal `App.User.{userId}`, para todos os usuários envolvidos na chamada.

6.4 NewRequest

`NewRequest` é uma notificação enviada quando um novo pedido de auditório é feito. É armazenada no banco de dados e enviada via *broadcasting* no canal `App.User.{userId}`, para todos os usuários do tipo secretário.

CAPÍTULO 7

Outros

7.1 Sistema de Notificações

O sistema de notificações funciona em tempo real graças à tecnologia WebSockets; em nossos testes, utilizamos o serviço Pusher. O componente de notificações faz uma requisição via ajax à rota `\notifications`, que retorna uma lista de notificações, que é adicionada ao menu de notificações. Quando uma notificação é criada, ou quando um evento do tipo `NotificationRead` é disparado, uma mensagem é enviada via *broadcasting* no canal `App.User.{userId}`, o que faz o componente de notificações gerar uma nova requisição e recarregar as notificações.

7.1.1 Componentes

Notification.vue

`Notification` é o componente responsável por gerenciar e exibir o menu de notificações, ele é adicionado na view `layouts.app`.

Na função `mounted()`, o seguinte pedaço de código é responsável por ouvir as mensagens que chegam via WebSockets e fazer a requisição ajax para recarregar as notificações:

```
function reloadNotifications() {
    $.get("/notifications", function (data, status) {
      if (status == 'success') {
        console.log('Notifications: Reloading notifications');
        self.unreadNotifications = data;
      }
    });
}

Echo.private(`App.User.${this.user_id}`)
  .notification((notification) => {
    console.log('Notifications: ' + notification.type);
    reloadNotifications();
```

(continues on next page)

(continuação da página anterior)

```
  })
  .listen('NotificationRead', (e) => {
    console.log('Notifications: App\\Events\\NotificationRead');
    reloadNotifications();
  });
```

NotificationItem.vue

`NotificationItem` é o componente que representa cada uma das notificações que são exibidas no menu de notificações.

7.2 Sistema de Chamadas

O Sistema de Chamadas permite que sejam criadas janelas de chat entre vários usuários. Assim como o Sistema de Notificações, funciona em tempo real graças à tecnologia WebSockets; O componente de chamada faz diversas requisições via ajax para atualizar as chamadas sem a necessidade de recarregar a página.

Quando uma mensagem é criada, um evento do tipo `MessageCreated` é disparado e enviado via *broadcasting* ao canal `App.Call.{callId}`. O componente de chamada, então, lida com a atualização das mensagens.

7.2.1 Componentes

Call.vue

Componente responsável por gerenciar uma chamada. Os métodos `refresh` e `load` lidam com as requisições ajax, enquanto o método `ListenOnEcho` ouve no canal `App.Call.{callId}`.

CallMember.vue

Componente que representa um usuário na lista de usuários na chamada.

CallMessage.vue

Componente que representa uma mensagem na lista de mensagens na chamada.

NewCall.vue

Componente para criação de chamadas.