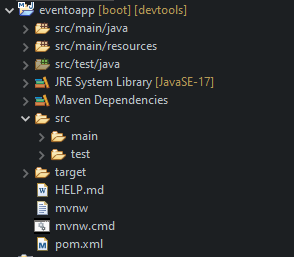
**Projeto – App de Eventos**

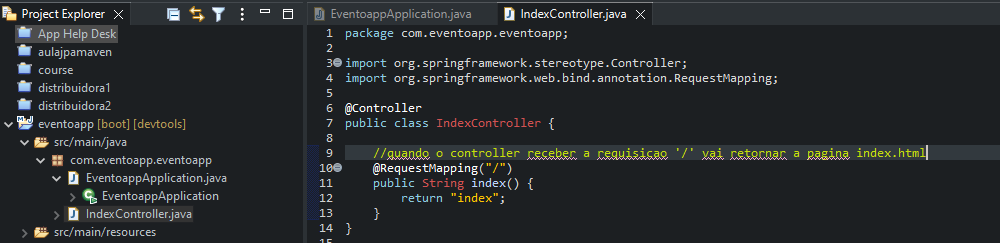
# Inicializando projeto



**Src/main/java ->** Onde ficam os controllers, models e as classes de configuração do banco e do spring.

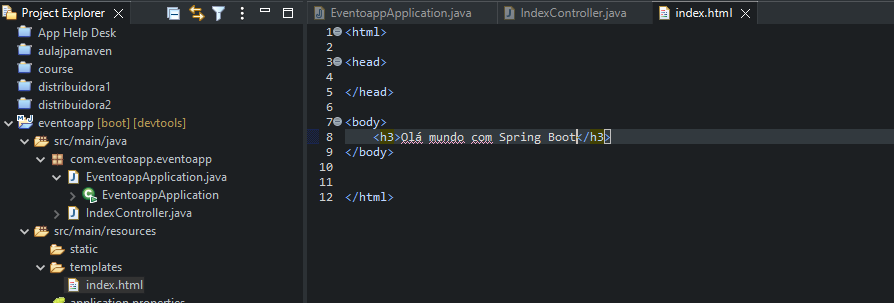
**Src/main/resources ->** Onde fica o html, css, javascript, etc. E também os templates que vão ser as views.

## Criando um controller e uma view para testar.

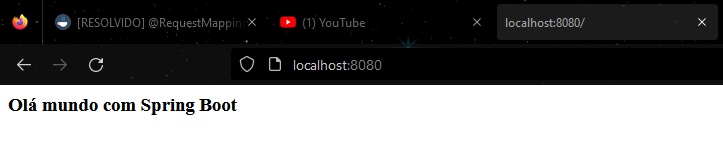


## Criando a view

Criar na pasta de resource/templates.

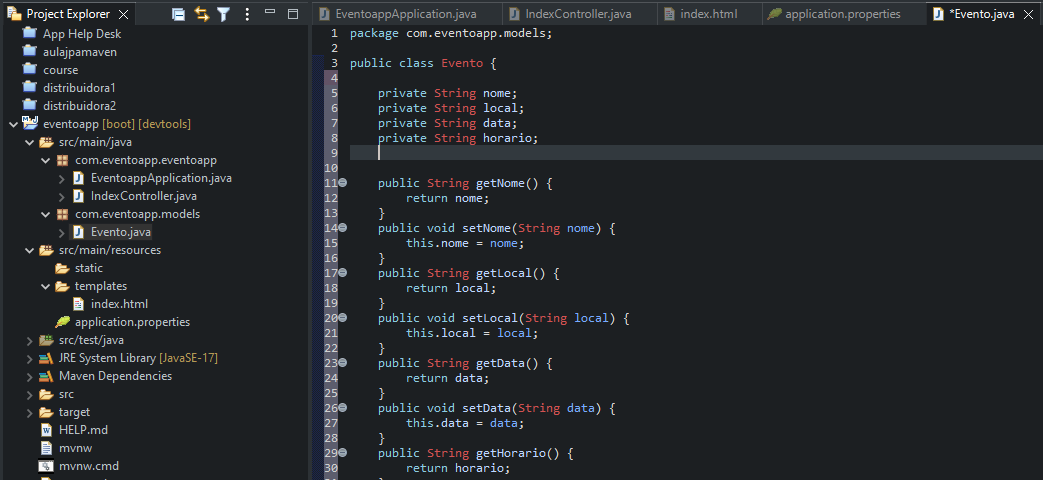


Agora ao acessar com servidor localhost já aparece a mensagem definida no template/view html.

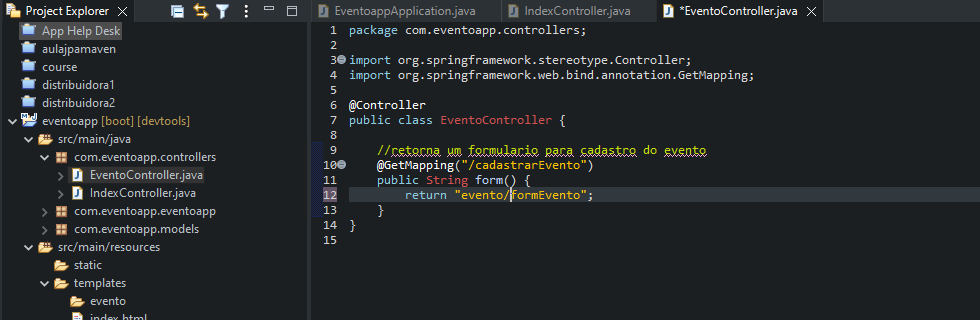


# Criando as camadas MVC (Model, View, Controller)

## Criando um model de evento



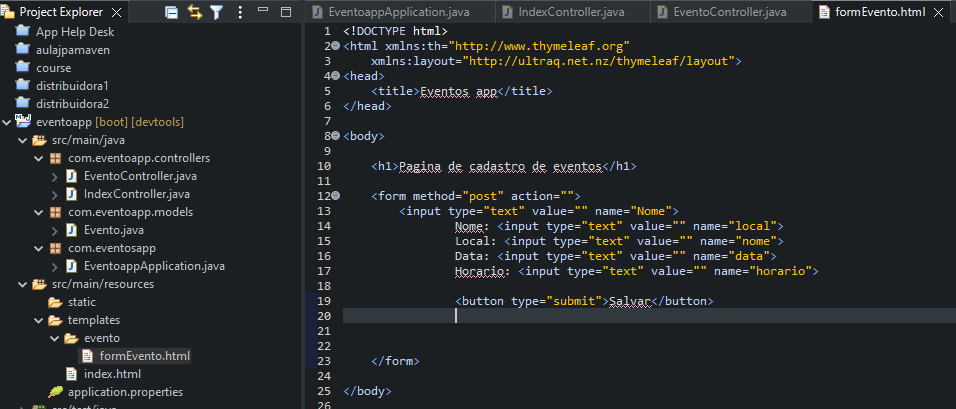
## Criando um controller de evento



**Obs: passado a pasta e o arquivo no return já que está em uma pasta dentro de template.**

## Criando a view formEvento

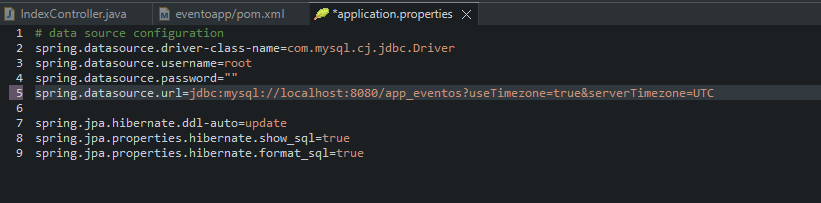
Src/main/resources/templates/evento



# Criando e configurando um banco de dados mysql

## Configurando BD e JPA

1 - Adicionada dependência do mysql e do jpa no pom.



2 - Adicionada essas configurações do banco e jpa no arquivo properties. Com isso configuramos o banco de forma mais simples.

## Configurando a tabela

Configurando a tabela Evento em models.

Unable to determine Dialect without JDBC metadata

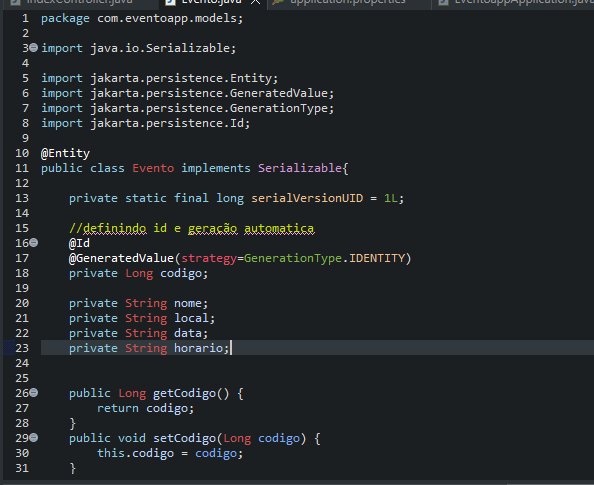
Para corrigir esse erro adicionado seguinte código ao arquivo properties:

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect

Para corrigir erro de JPA e Hibernate não criar as tabelas no banco usar na classe principal:

@EntityScan(basePackages = {"com.eventoapp.models"})

Tabela criada em models e no banco:

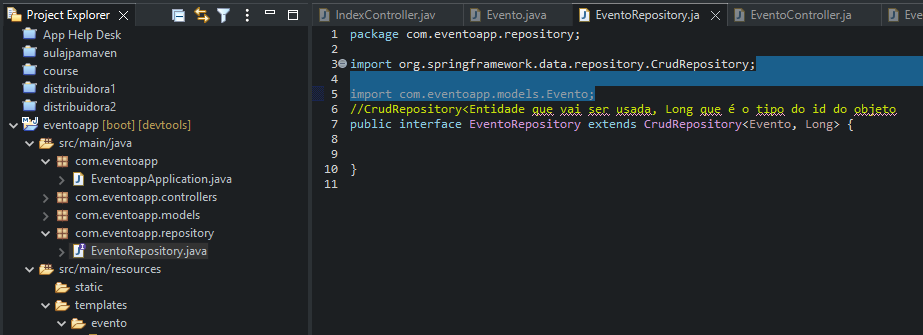


# Salvando dados do formulário no banco de dados

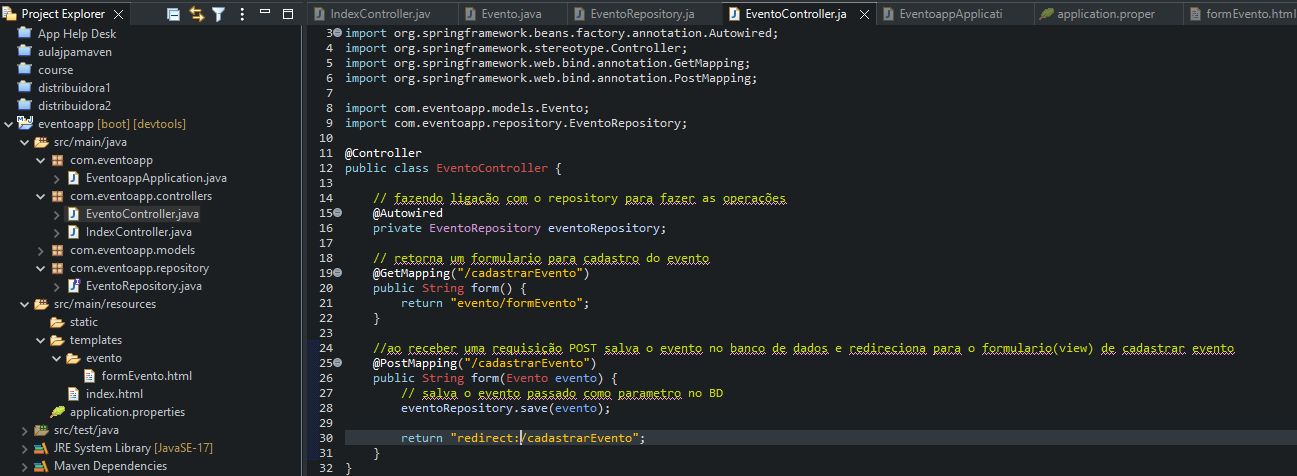
Criado package repositories e interface que vai implementar o JPA. Usada para as operações no banco de dados. Vai extender CrudRepositories para operações básicas.

Com.eventosapp.repository/EventoRepository

@Autowired -> faz uma injeção de dependência, cada vez que precisarmos usar a interface vai ser criada uma nova instancia automaticamente.



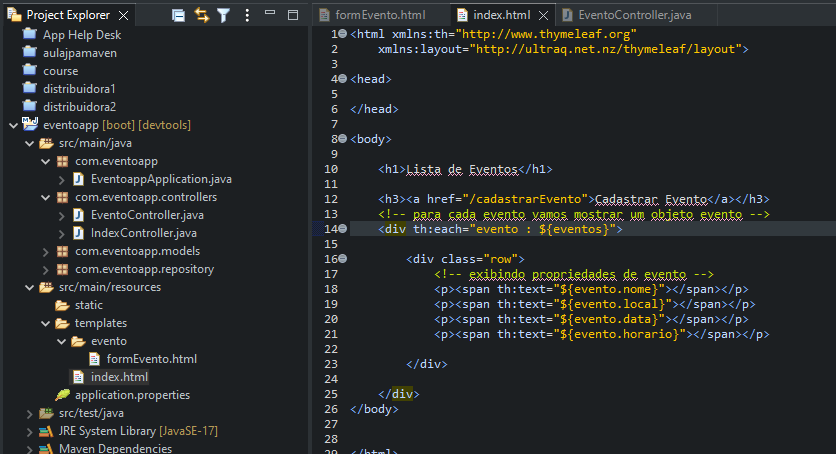
Interface EventoRepository.

Classe EventoController onde são feitas as operações de Evento.

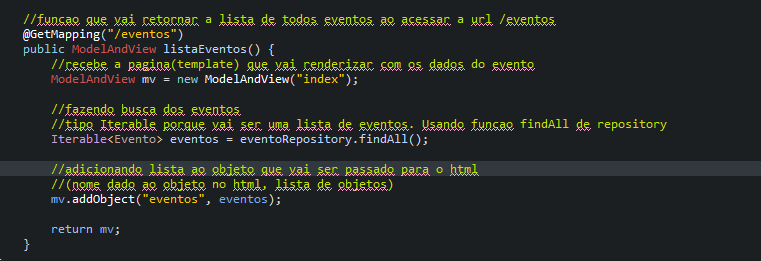
**Com isso ao clicar em salvar no formulário html os dados já são salvos no banco.**

# Realizando buscas no banco de dados

Para exibir os eventos já cadastrados vamos primeiro criar uma view.



Agora vamos configurar a função no controller. EventoController

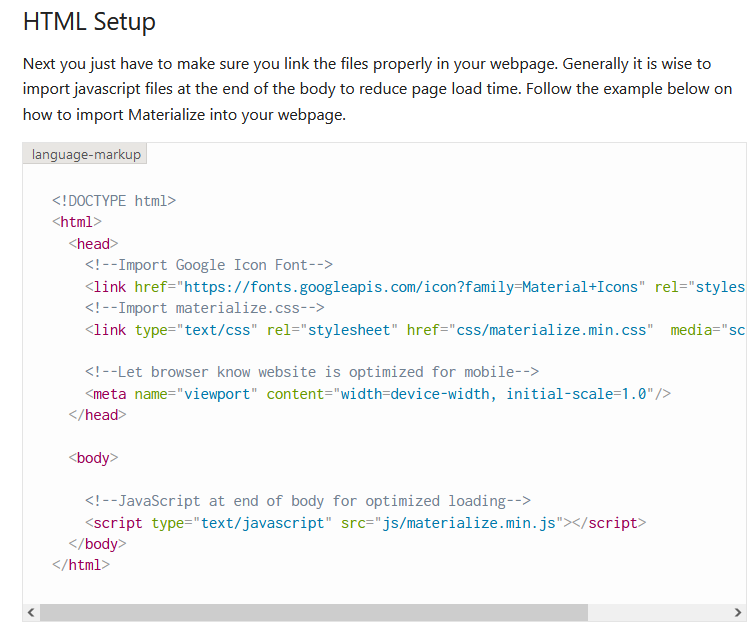


Ao acessar a url local <http://localhost:8080/eventos> temos a listagem de dados exibida.



# Utilizando o Materialize css na aplicação

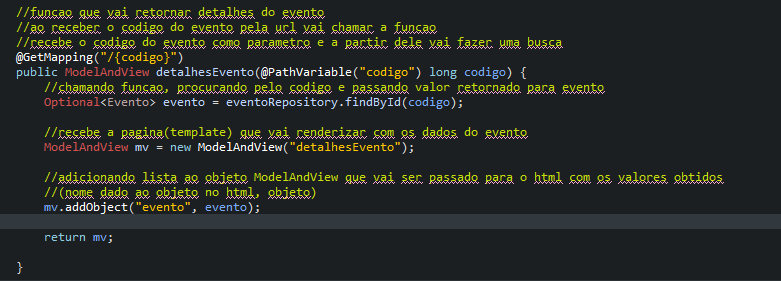
Baixado arquivo materialize.css e extraído na pasta **C:\Users\Leo\eclipse-workspace\eventoapp\src\main\resources\static** para usarmos no projeto.

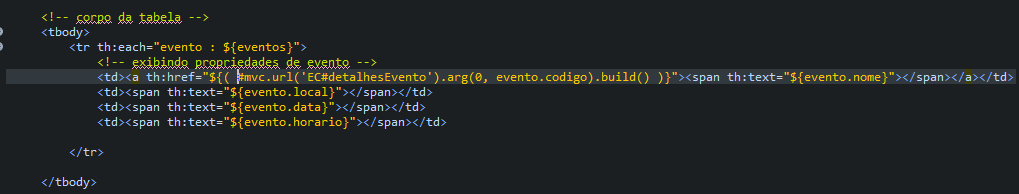


Feitas as configurações em todos templates da aplicação.

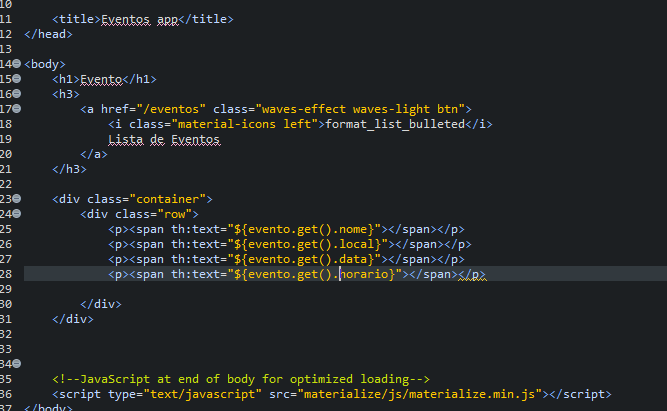
# Criando requisições especificas com a anotação @PathVariable

Criando função que retorna uma pagina com os detalhes de cada evento. Usando o findById.



Criando um link no html, ao passar cursor por cima vai ser possível clicar no nome do evento e abrir os detalhes.

Agora vamos criar o template de **detalhesEvento.html.**



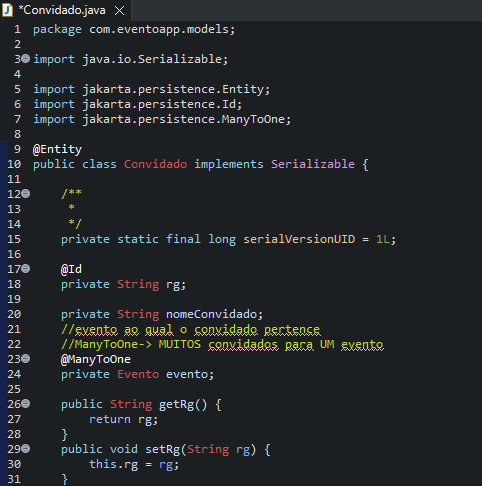
Com isso conseguimos acessar os atributos do objeto especifico na página.

# Relacionando entidades utilizando @OneToMany e @ManyToOne

Criando o model convidados.

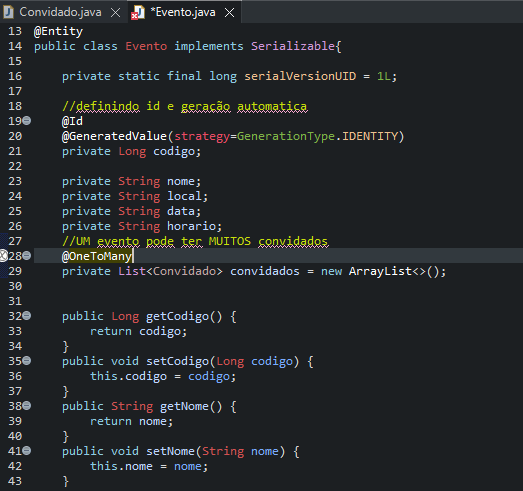
Um convidado pode ter apenas um evento, já o evento pode ter muitos convidados, vamos fazer essa relação primeiro em Convidado.

ManyToOne -> muitos (convidados, classe atual) para um (evento, classe separada).



Agora vamos fazer a relação em Evento.

OneToMany -> um (evento, classe atual) pode ter muitos (convidados, classe separada)



**Com isso a relação entre as duas tabelas está feita.**

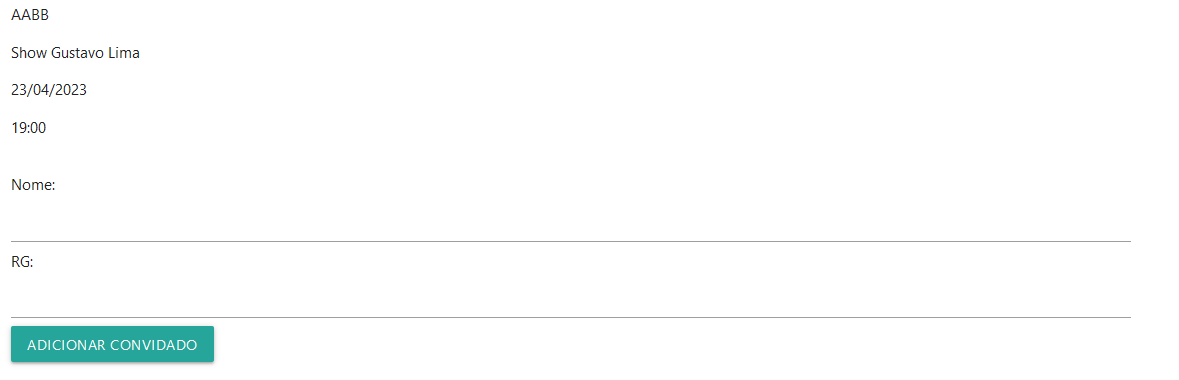
Criando formulário em detalhesEvento para cadastrar os convidados.

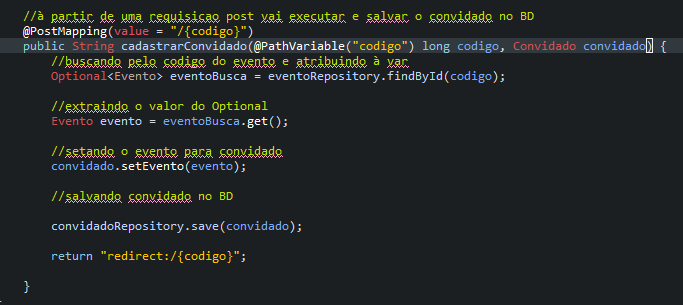


# Salvando um dado de uma entidade no banco relacionado a outra entidade

Vamos criar um método para receber via Post um formulário para adicionar convidados ao evento. Recebe código e o convidado por parâmetro.

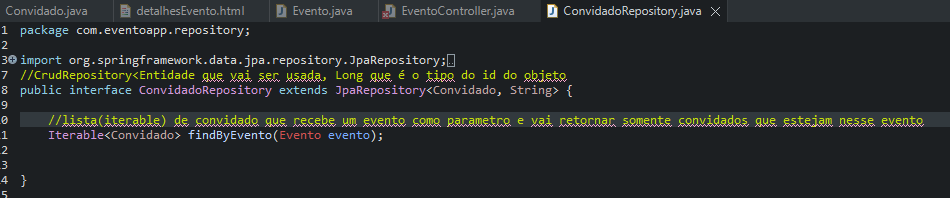
Ao clicar no botão de **adicionar convidado** no evento vai acionar o envio do formulário por **POST** e com isso chamar o método **cadastrarConvidado** que seta um **evento** em **convidado** com base no **código** obtido anteriormente e **salva** o **convidado** no BD.



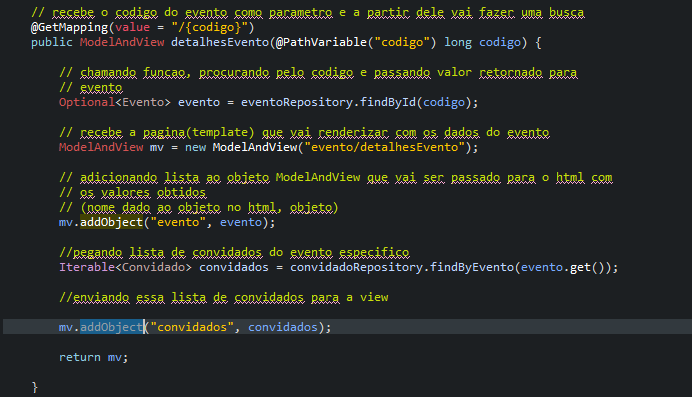


# Trazendo lista de convidados do evento

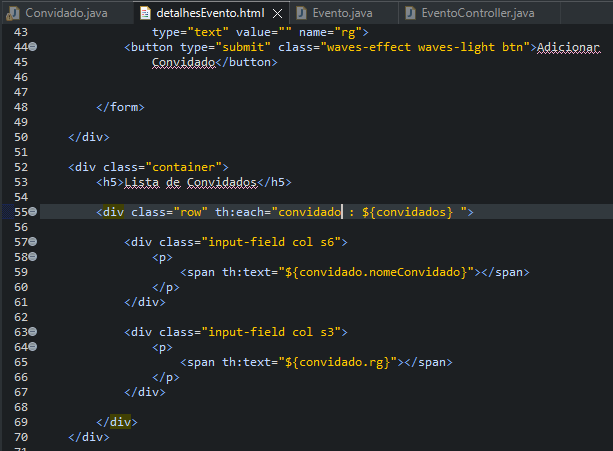
Criando método customizado no repository para trazer apenas convidados do evento.



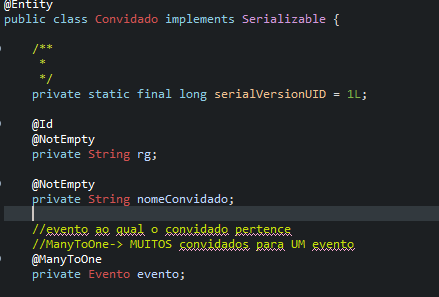
Fazendo as alterações na função de EventoController(que traz os detalhes do evento) para trazer também a lista de convidados.



Fazendo alterações na view para trazer os dados do controller.



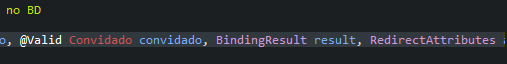
# Fazendo validação de dados para serem salvos no banco



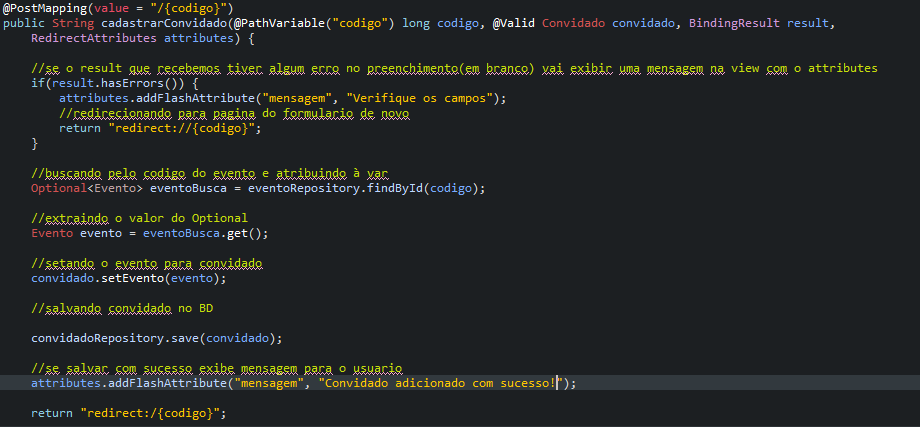
Adicionando notação @NotEmpty para rg e nomeCondivdado em convidado, assim o banco não vai aceitar valores vazios.



Adicionando @Valid no convidado do parâmetro na função de cadastrarConvidado.



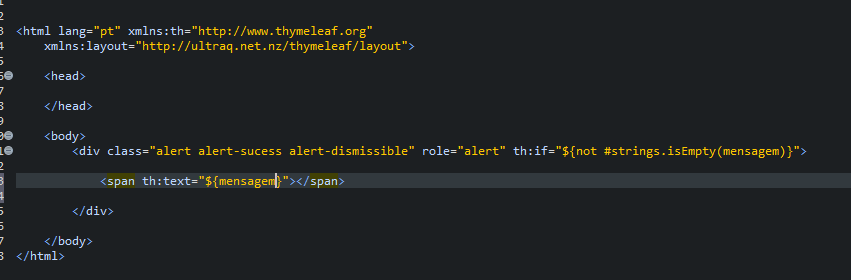
Recebe também esses parâmetros.

Fazendo verificação dos valores recebidos pelo formulário.

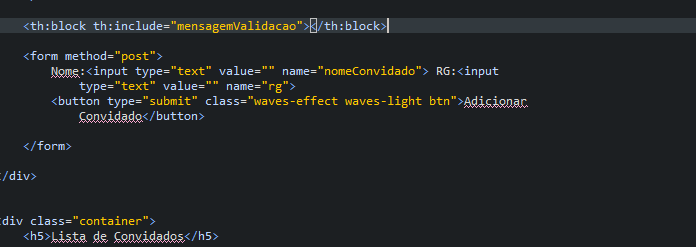
Incluindo mensagem no html.

Templates > New File: mensagemValidacao.html

Vai verificar a mensagem que passamos com atributo e mostrar ela.



Agora temos que incluir esse arquivo html na pagina que temos o formulário de cadastro de convidados(evento/detalhesEvento.html). Para isso vamos usar a tag **th:block** do **thymeleaf.**



**Após isso feito o mesmo para evento.**

# Fazendo ajustes no html com materialize

Ajustando todos templates e criando arquivo css para aplicar formatações.

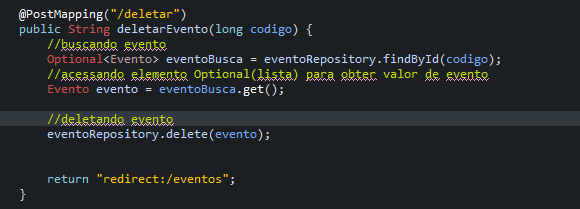
# Deletando elementos do banco de dados

Criando método que remove evento.

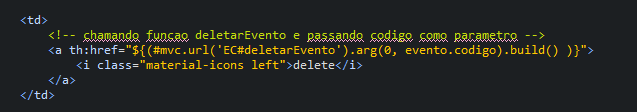
@OneToMany(mappedBy = "evento", cascade = CascadeType.ALL)

private List<Convidado> convidados = new ArrayList<>();

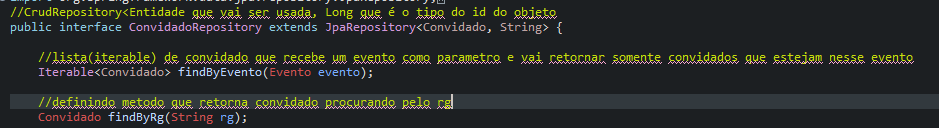
Adicionado essa configuração na declaração da relação entre evento e convidados para que ao remover evento os convidados sejam removidos também.



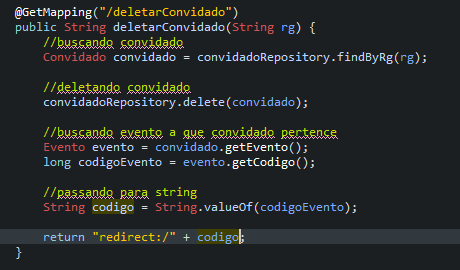
Criando botão no template para acionar o controller deletarEvento.



Criando função de deletar para convidado.

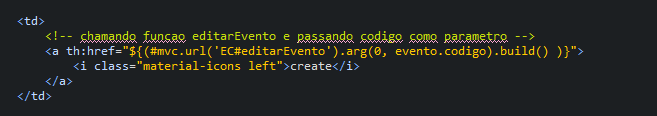
Primeiro criar uma função na interface convidadoRepository para retornar o convidado por rg(String).

Criando método no controller.

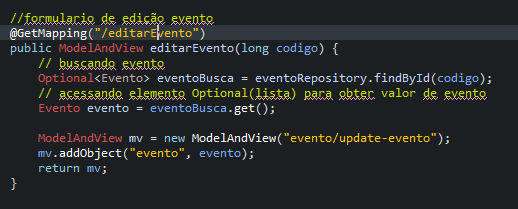


Criando botão no template.

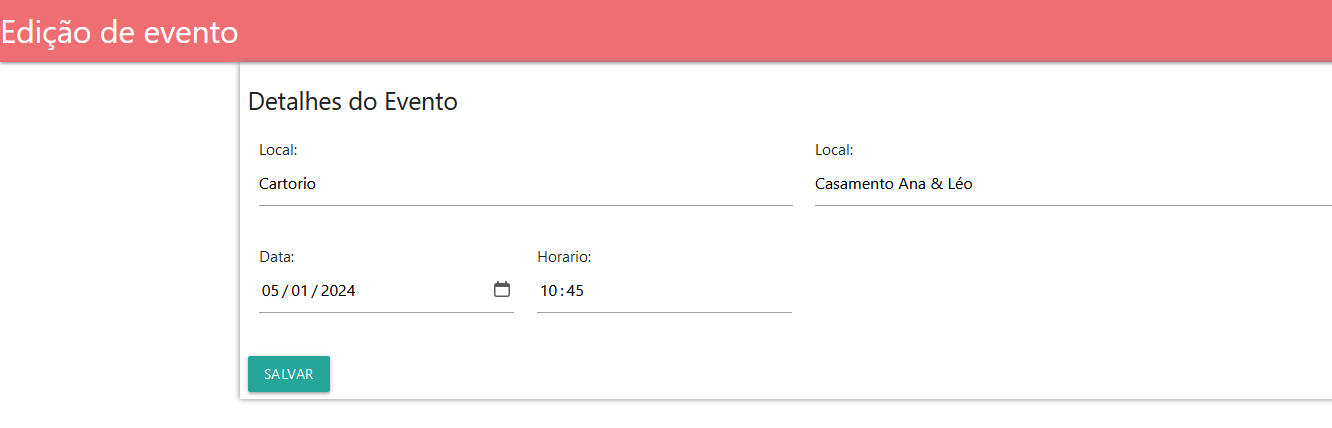
# Editando elementos (Update)



Adicionando HTML para chamar a view de editar evento passando o código do evento como parâmetro.



Ao clicar no botão no HTML passa um GET na url e redireciona para a função de retornar a view.



Ao clicar em salvar vai chamar a função /editar-evento (POST) definida no EventoController.

