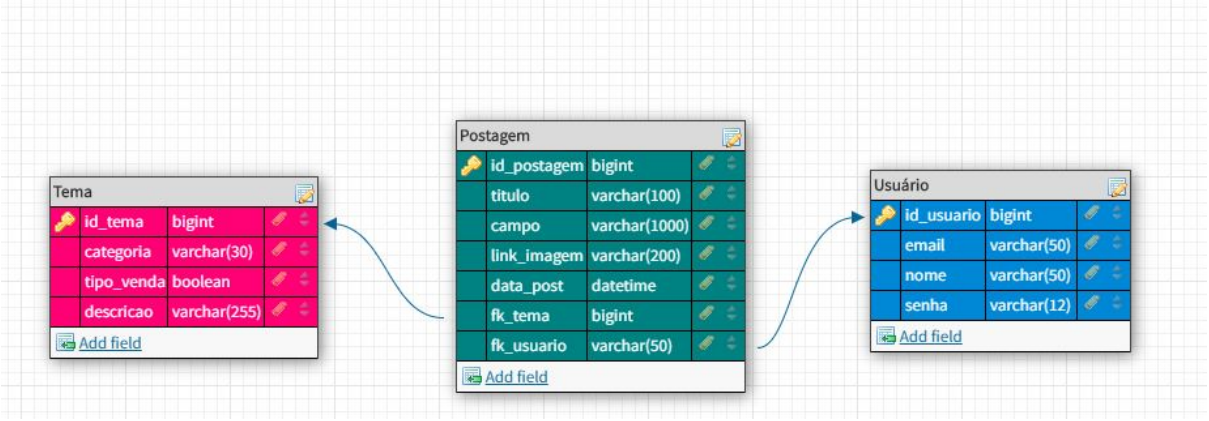
Grupo 1 I - Mei

Integrantes: Cinthia Tengan, Lucas Corrêa, João Paulo Oliveira, Sthefany Silva, Sérgio Ruiz.

Descrição das tabelas e atributos do modelo DER.



De acordo com as instruções propostas pelo instrutor, foram criadas 3 tabelas com seus respectivos atributos.

- **Tema**

id_tema: chave primária para poder identificar o tema escolhido; categoria: classificação do tema;

categoria: classificar o tema;

tipo_venda: de acordo com a regra de negócio, escolhemos que devem ter dois tipos produtos e serviços, e neste caso usamos um valor booleano no qual 0 representa produto e 1 o serviço;

descricao: descrever o tema selecionado com mais detalhe;

- **Usuario**

id_usuario: chave primária para identificar o usuário.

email: email do usuario.

Nome: nome do usuario completo;

senha: senha de acesso do usuário;

Nota: O atributo id_usuario foi criado posteriormente, pois ao utilizarmos o spring precisamos de uma chave primária com números, e não letras como havíamos escolhido o email anteriormente.

- **Postagem**

id_postagem: chave primária para identificar a postagem;

fk_email: chave estrangeira da tabela usuario para poder identificar o usuário que fez a postagem;

fk_tema: chave estrangeira da tabela tema para poder identificar o tema escolhido pelo o usuário na postagem;

titulo: titulo da postagem;

campo: texto principal da postagem do usuário;

link_imagem: armazena a ‘url’ da imagem e ‘linka’ com a postagem;

data_post: data da postagem feita do usuário.

Criado o arquivo Aplication Properties:

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/db_I_mei?createDatabaseIfNotExist=true&serverTimezone=UTC&useSSl=false

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.show-sql=true

// Com esses comandos spring, o banco de dados atualiza automatico, e se o banco não existe ele é criado.

Criação do pacote Model , com a classe tema

@Entity // essa classe será definida como uma entidade do jpa hibernate.

@Table(name = "tema") // dentro do banco de dados uma tabela será criada com o nome de tema.

//criação da classe tema:

public class Tema {

//Dividindo os atributos para explicar as suas anotações:

@Id // Identifica o atributo id como uma primary key.

@Column(name = "id_tema") // renomeia a coluna como id_tema no banco de dados ao invés do atributo id no java.

@GeneratedValue(strategy = GenerationType.IDENTITY) // diz que esse atributo será auto incrementado.

private long id;

@NotNull // anotação para dizer que o valor não pode ser nulo

@Size(max = 30) // tamanho máximo do atributo

private String categoria;

@Column(name = "tipo_venda") //renomeia a coluna como tipo_venda no banco de dados ao invés do atributo tipoVenda no java.

@NotNull // anotação para dizer que o valor não pode ser nulo

private boolean tipoVenda;

@NotNull // anotação para dizer que o valor não pode ser nulo

private String descricao;

Anotações utilizadas nas classes Controller:

Anotações utilizadas antes da classe:

@RestController // Anotação utilizada para tornar a classe um controller

@RequestMapping("/postagem") //caminho para chamar o controller

@CrossOrigin(origins = "*", allowedHeaders = "*") //Anotação para permitir solicitações de origem cruzada em classes de manipuladores específicos e/ou métodos de manipulador. Processado se um HandlerMappings apropriado estiver configurado

Anotações dos métodos:

@Autowired //Anotação para ligar o controller na interface do repository. Ele "assina o contrato".

private PostagemRepository repository; // criado um objeto do tipo PostagemRepository.

@GetMapping //anotação para mapear um pedido http de pedido com os seguintes requisitos dentro do método

```
public ResponseEntity<List<Postagem>> findAllPostagem() {
        return ResponseEntity.ok(repository.findAll());
}
```
//método para buscar todos os itens de postagem

```
@GetMapping("/{id}")
public ResponseEntity<Postagem> findByIdPostagem(@PathVariable long id) {
```

```
        return repository.findById(id).map(resp -> ResponseEntity.ok(resp)).orElse(ResponseEntity.notFound().build());
    } //método para buscar os itens de postagem com a adição do id na busca

    @PostMapping //método para postar um item novo na tabela postagem.
    public ResponseEntity<Postagem> postPostagem(@RequestBody Postagem postagem) {
        return ResponseEntity.status(HttpStatus.CREATED).body(repository.save(postagem));
    }

    @PutMapping //método para atualizar um item existente na tabela postagem.
    public ResponseEntity<Postagem> putPostagem(@RequestBody Postagem postagem) {
        return ResponseEntity.status(HttpStatus.OK).body(repository.save(postagem));
    }

    @DeleteMapping("/{id}") //método para deletar um item da tabela postagem através do id do item da tabela postagem.
    public void deletePostagem(@PathVariable long id) {
        repository.deleteById(id);
    }
}
```

CRUD Definição

CRUD é o acrônimo da expressão do idioma Inglês, Create (Criação), Read (Consulta), Update (Atualização) e Delete (Destruição). Este acrônimo é comumente utilizado para definir as quatro operações básicas usadas em Banco de Dados Relacionais.

(16/12/20) Criação do pacote Repository com a interface TemaRepository e do pacote Controller com a classe TemaController.

(17/12/20) Criação do CRUD das tabelas Postagem e Usuário.

Relacionamento entre tabelas já criadas.

Criação do pacote Model , com a classe Postagem