FACULDADE SENAC GOIAS

GESTÃO TECNOLOGIA DA INFORMAÇÃO

SOFTWARE ECOMMERCE

PROF: ROUSSIAN

GOIÂNIA

2020

FACULDADE SENAC GOIAS

GESTÃO TECNOLOGIA DA INFORMAÇÃO

VINICIUS LOPES SILVA

LUCAS EDUARDO RODRIGUES

FABRICIO MOREIRA MACHADO

ELSON CRISTINO FARIAS

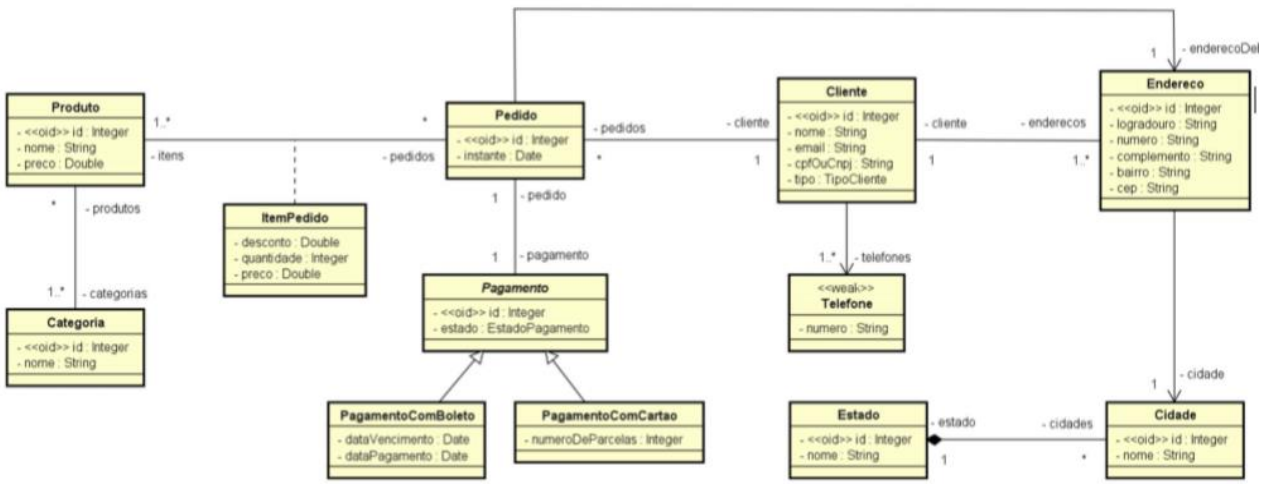
GOIÂNIA

2020

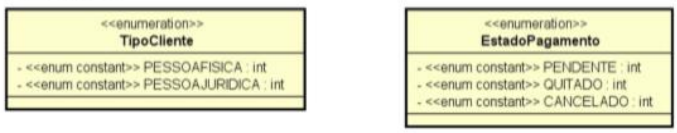
INTRODUÇÃO

Foi criado um software de comercio eletrônico onde possui funções de compra , alteração , cadastro que é feito somente pelo administrador, ele possui uma funcionalidade de confirmação de compra enviado por e-mail.

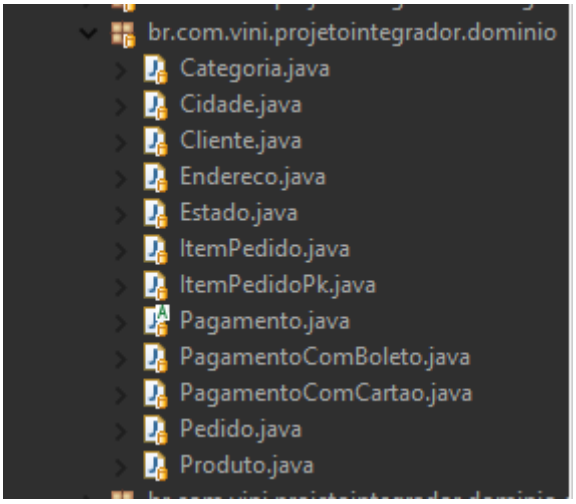
A linguagem desse software é feito em java , com as ferramentas de mapeamento relacional, como hibernate, banco de dados para auxiliar para criação do banco de Dados , esse software possui ferramentas de segurança de autenticação ao acessar o app , ele permite selecionar o tipo de pessoa se é jurídica ou pessoa física, e também, foi baseado em um modelo relacional comforme a imagem abaixo.



Enumerações:



Classe criadas

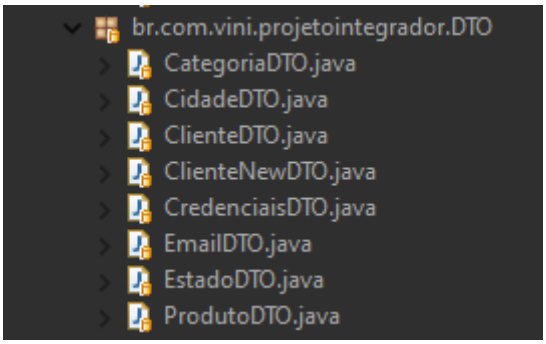


Atributos criados na classes principais:

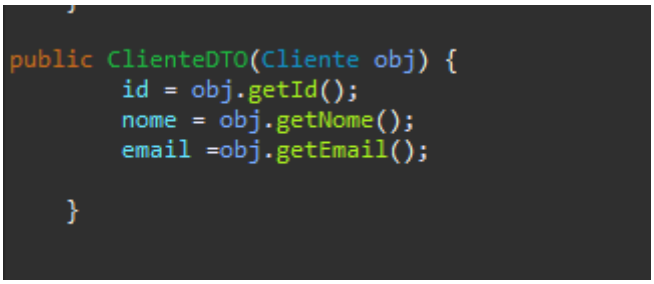
- Atributos básicos
- Associações (inicie as coleções)
- Construtores (não inclua coleções no construtor com parâmetros) o Getters e setters
- hashCode e equals (implementação padrão: somente id)
- Serializable (padrão: 1L)

Link :https://github.com/vinpc/Back-end-projeto/blob/master/src/main/java/br/com/vini/projetointegrador/dominio/Cliente.java

Classes DTOS



O DTO é uma das alternativas para projeção de dados, você pode projetar diretamente da Entity como vimos no início. Particularmente gosto do DTO por dar liberdade nessa projeção de dados. Uma entidade é algo que deveria mudar pouco, ainda mais em casos que ela está fortemente acoplada com banco de dados relacional.



```
public ClienteDTO(Cliente obj) {
        id = obj.getId();
        nome = obj.getNome();
        email =obj.getEmail();

    }
```

Com esse construtor foi gerado um Objeto chamado Obj apartir da classe cliente onde pegamos os get e usamos como objetos id,nome,email para melhoria na projeção de dados

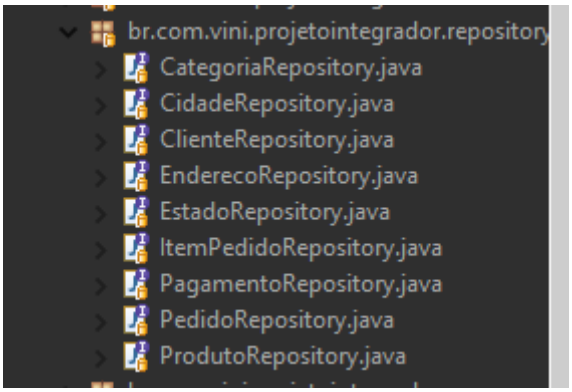
Através das classes DTO foram criadas algumas regras definidas

A partir das anotações @NoEmpity @Length

```
private Integer id;

@NotEmpty(message = "preenchimento obrigatorio")
@Length(min= 5, max=120, message="O Tamanho deve ser entre 5 a 120 caracteres")
private String nome;

@NotEmpty(message = "preenchimento obrigatorio")
@Email(message="email invalido")
private String email;

@NotEmpty(message = "preenchimento obrigatorio")
private String senha;

public ClienteDTO() {
```

Repositorys

A implementação de uma camada de acesso a dados de um aplicativo é complicada há bastante tempo. Muito código clichê teve que ser escrito. As classes de domínio eram anêmicas e não foram projetadas de maneira real orientada a objetos ou orientada a domínio. O uso dessas duas tecnologias facilita muito a vida dos desenvolvedores em relação à persistência do modelo de domínio avançado. No entanto, a quantidade de código padrão para implementar repositórios especialmente ainda é bastante alta. Portanto, o objetivo da abstração do repositório do Spring Data é reduzir significativamente o esforço para implementar camadas de acesso a dados para vários armazenamentos de persistência. Os capítulos a seguir apresentarão os principais conceitos e interfaces dos repositórios Spring Data em geral, para obter informações detalhadas sobre os recursos específicos de uma loja específica, consulte os capítulos posteriores deste documento.



Operações de CRUD e Casos de Uso

Objetivo geral:

Implementar operações de CRUD e de casos de uso conforme boas práticas de Engenharia de Software.

Competências:

 Implementar requisições POST, GET, PUT e DELETE para inserir, obter, atualizar e deletar entidades, respectivamente, seguindo boas práticas REST e de desenvolvimento em camadas.

Trabalhar com DTO (Data Transfer Object)

Trabalhar com paginação de dados

 Trabalhar com validação de dados com Bean Validation (javax.validation)

 Criar validações customizadas

 Fazer tratamento adequado de exceções (incluindo integridade referencial e validação)

 Efetuar consultas personalizadas ao banco de dados

Inserindo novo Cliente com POST

{

"nome" : "João da Silva",

"email" : "joao@gmail.com",

"cpfOuCnpj" : "39044683756",

"tipo" : 1, "telefone1" : "997723874",

"telefone2" : "32547698",

"logradouro" : "Rua das Acácias",

"numero" : "345",

"complemento" : "Apto 302",

"cep" : "38746928",

"cidadeId" : 2

}

Exemplo da requisição GET

Spring Security

Spring Security tem o foco em tornar a parte de autenticação e autorização uma coisa simples de fazer. Ele tem uma variedade muito grande de opções e ainda é bastante extensível.

Com algumas poucas configurações já podemos ter uma autenticação via banco de dados, LDAP ou mesmo por memória. Sem falar nas várias integrações que ele já suporta e na possibilidade de criar as suas próprias.

Quanto a autorização, ele é bem flexível também. Através das permissões que atribuímos aos usuários autenticados, podemos proteger as requisições web (como as telas do nosso sistema, por exemplo), a simples invocação de um método e até a instância de um objeto.

Autenticação e Autorização com tokens JWT

- Compreender o mecanismo de funcionamento do Spring Security
- Implementar autenticação e autorização com JWT
- Controlar conteúdo e acesso aos endpoint

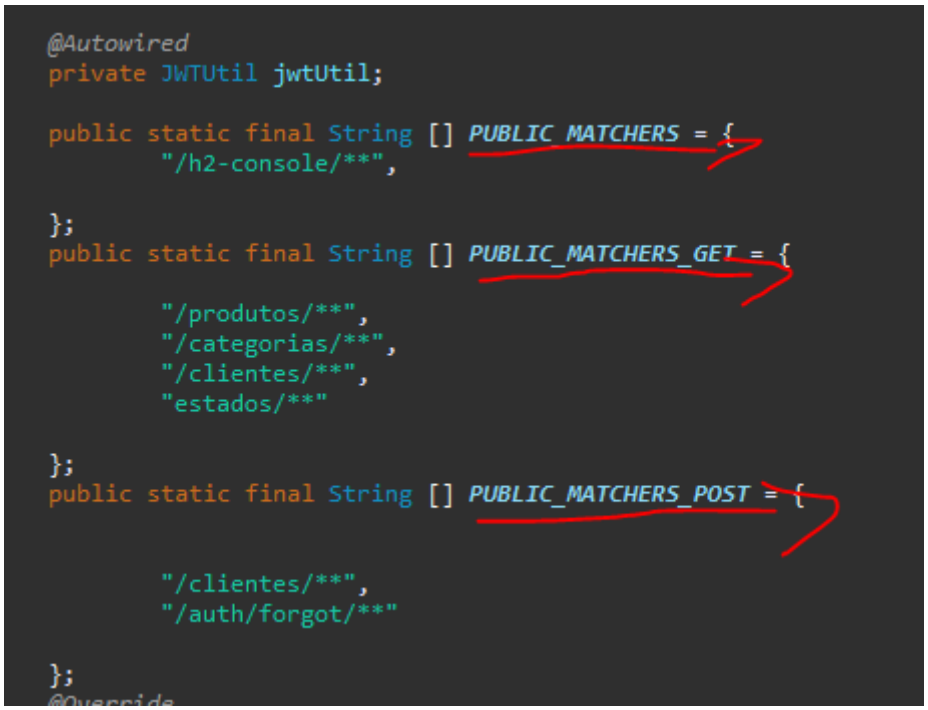
Configuração inicial do Spring Security

- Incluir as dependências no pom.xml
- Criar uma classe de configuração SecurityConfig para definir as configurações de segurança
- Esta classe deve herdar de WebSecurityConfigurerAdapter
- Definir as configurações básicas das URL's que necessitam ou não de autenticação/autorização

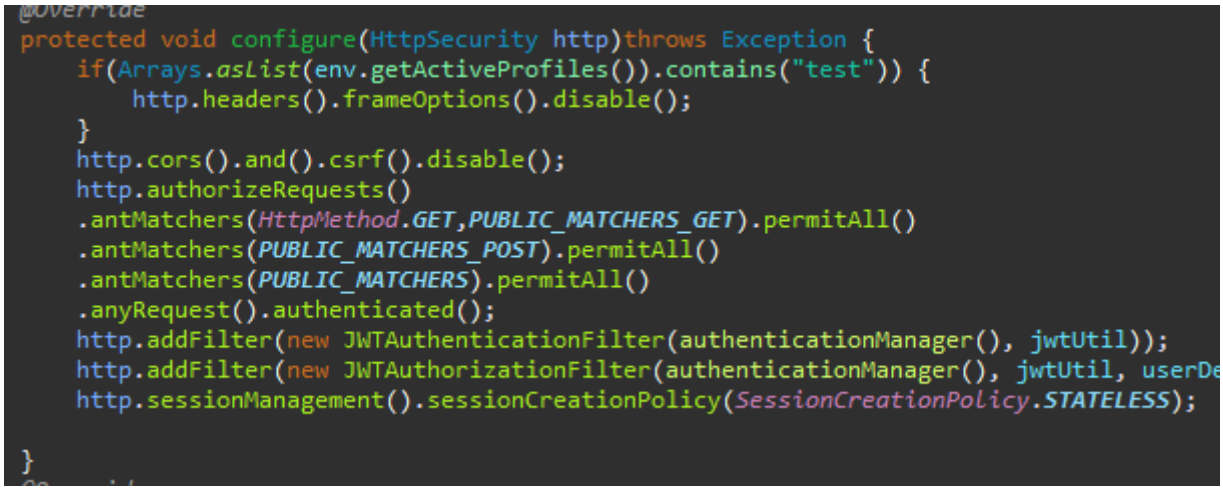
Classe SecurityConfig

A classe Security config é uma classe onde é definida as regras de acesso as paginas principais web , ela é fundamental para a segurança de seu software.

A classe estática PUBLIC_MATCHERS é fundamental para criar regras para mostrar informações no endpoint onde realiza o filtro de segurança de acesso é fundamental para criar regras nas requisições GET , PUT ,POST,DELETE.



No código abaixo utilizamos um conjunto de regras e acessos de cada endpoint no http.



```
@Override
protected void configure(HttpSecurity http)throws Exception {
    if(Arrays.asList(env.getActiveProfiles()).contains("test")) {
        http.headers().frameOptions().disable();
    }
    http.cors().and().csrf().disable();
    http.authorizeRequests()
    .antMatchers(HttpMethod.GET,PUBLIC_MATCHERS_GET).permitAll()
    .antMatchers(PUBLIC_MATCHERS_POST).permitAll()
    .antMatchers(PUBLIC_MATCHERS).permitAll()
    .anyRequest().authenticated();
    http.addFilter(new JWTAuthenticationFilter(authenticationManager(), jwtUtil));
    http.addFilter(new JWTAuthorizationFilter(authenticationManager(), jwtUtil, userDe
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

}
```

Adicionando senha a Cliente

- Incluir um Bean de BCryptPasswordEncoder no arquivo de configuração
- Incluir um atributo senha na classe Cliente
- Atualizar ClienteNewDTO
- Atualizar ClienteService
- Atualizar DBService

Salvando perfis de usuário na base de dados

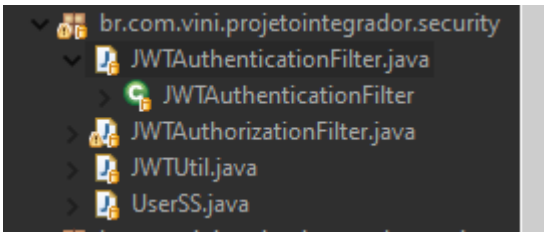
- Criar o tipo enumerado Perfil (CLIENTE, ADMIN)
- Implementar na classe Cliente: o Um atributo correspondente aos perfis do usuário a serem armazenados na base de dados
- □ Usar @ElementCollection(fetch=FetchType.EAGER) o Disponibilizar os métodos: □ public Set<Perfil> getPerfis() □ public void addPerfil(Perfil perfil)
- Incluir perfil padrão (CLIENTE) na instanciação de Cliente
- Incluir mais um cliente com perfil ADMIN na carga inicial da base de dados (DBService)

Implementando autenticação e geração do token JWT

Criar classe de usuário conforme contrato do Spring Security (implements UserDetails)

- Criar classe de serviço conforme contrato do Spring Security (implements UserDetailsService)
- Em SecurityConfig, sobrescrever o método: public void configure(AuthenticationManagerBuilder auth)
- Criar a classe CredenciaisDTO (usuário e senha)
- Incluir as propriedades de JWT (segredo e tempo de expiração) em application.properties
- Criar uma classe JWTUtil (@Component) com a operação String generateToken(String username)
- Criar um filtro de autenticação
- Em SecurityConfig, registrar o filtro de autenticação

Confira a imagem abaixo:



Observações importantes :

@Component	Anotação genérica para qualquer componente gerenciado pelo Spring. Esta anotação faz com que o bean registrado no Spring possa ser utilizado em qualquer bean, seja ele um serviço, um DAO, um controller, etc. No nosso exemplo, ele será responsável por um Bean que representa uma entidade.
@Repository	Anotação que serve para definir uma classe como pertencente à camada de persistência.
@Service	Anotação que serve para definir uma classe como pertencente à camada de Serviço da aplicação.
@Autowired	A anotação @ Autowired fornece controle sobre onde e como a ligação entre os beans deve ser realizada. Pode ser usado para em métodos setter, no construtor, em uma propriedade ou métodos com nomes arbitrários e / ou vários argumentos.

Bônus:

Seviço de e-mail: em spring

- Adicionar a dependência no POM.XML
- Remetente e destinatário default no application.properties
- Criar a interface EmailService (padrão Strategy)
- Criar a classe abstrata AbstractEmailService
- Criar método prepareSimpleMailMessageFromPedido
- Sobrescrever o método sendOrderConfirmationEmail (padrão Template Method) Implementar o MockEmailService
- Em TestConfig, criar um método @Bean EmailService que retorna uma instância de MockEmailService

Arquivo de configuração de email :

```
spring.mail.host=smtp.gmail.com
spring.mail.username=
spring.mail.password=
spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.socketFactory.port = 465
spring.mail.properties.mail.smtp.socketFactory.class = javax.net.ssl.SSLSocketFactory
spring.mail.properties.mail.smtp.socketFactory.fallback = false
spring.mail.properties.mail.smtp.starttls.enable = true
spring.mail.properties.mail.smtp.ssl.enable = true
```

Link principal do projeto

https://github.com/vinpc/Back-end-projeto