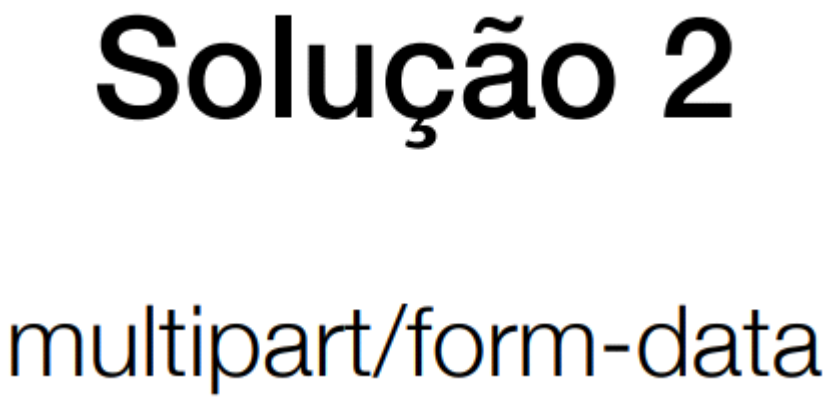
14.1. Conhecendo soluções para upload de arquivos em REST APIs

domingo, 2 de abril de 2023 09:42



14.2. Implementando upload de arquivo com multipart/form-data

domingo, 2 de abril de 2023 10:27

```java
@RestController
@RequestMapping("restaurantes/{restauranteId}/produtos/{produtoId}/foto")
public class RestauranteProdutoFotoController {

    @PutMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public void atualizarFoto(@PathVariable Long restauranteId,
                              @PathVariable Long produtoId,
                              FotoProdutoInput fotoProdutoInput){
        final String originalFilename = UUID.randomUUID().toString() + "_" + fotoProdutoInput.getFile().getOriginalFilename();

        final Path path = Path.of( first: "C:\\Users\\Guto1\\Documents\\dev\\" + originalFilename);

        System.out.println(path);
        System.out.println( fotoProdutoInput.getFile().getContentType());

        try {
            fotoProdutoInput.getFile().transferTo(path);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

```java
@Getter
@Setter
public class FotoProdutoInput {

    @NotNull
    @FileSize(max = "100KB")
    private MultipartFile file;

    @NotBlank
    private String descricao;

}
```

Atributos a propriedade consumes com o valor MULTIPART_FORM_DATA_VALUE para que o endpoint só aceite o tipo form-data

14.3. Validando o tamanho máximo do arquivo

domingo, 2 de abril de 2023 11:34

- Validar com as constraints do Bean Validation as propriedades do arquivo input da requisição

```java
@Getter
@Setter
public class FotoProdutoInput {

    @NotNull
    @FileSize(max = "100KB")
    private MultipartFile file;

    @NotBlank
    private String descricao;

}
```

Por padrão o Spring utiliza tamanhos fixos para requisições e para arquivos multipart
#spring.servlet.multipart.max-file-size=20KB
#spring.servlet.multipart.max-request-size=20MB

- Criar validações customizadas do Bean Validation

```java
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE, ElementType.CONSTRUCTOR,
        ElementType.PARAMETER, ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {FileSizeValidator.class})
public @interface FileSize {
    String message() default "tamanho do arquivo é inválido";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    2 usages
    String max();
}
```

```java
public class FileSizeValidator implements ConstraintValidator<FileSize, MultipartFile> {

    2 usages
    private DataSize dataSize;

    @Override
    public void initialize(FileSize constraintAnnotation) {
        this.dataSize = DataSize.parse(constraintAnnotation.max());
    }

    @Override
    public boolean isValid(MultipartFile file, ConstraintValidatorContext constraintValidatorContext) {

        /*O tamanho do arquivo da requisição é em Bytes e datasize guarda o valor limite especificado na anotação */
        return file == null || file.getSize() <= dataSize.toBytes();
    }
}
```

```properties
#14.3. Validando o tamanho máximo do arquivo
fotoProdutoInput.file=Arquivo da foto
fotoProdutoInput.descricao=Descrição da foto
FileSize.fotoProdutoInput.file=Foto deve ter o tamanho máximo de {1}
```

14.4. Desafio: Validando o content type do arquivo

domingo, 2 de abril de 2023 13:56

- Criar uma anotação customizada

```
@Getter
@Setter
public class FotoProdutoInput {

    @NotNull
    @FileSize(max = "500KB")
    @FileContentType(allowed = {MediaType.IMAGE_JPEG_VALUE, MediaType.IMAGE_PNG_VALUE})
    private MultipartFile file;

    @NotBlank
    private String descricao;

}
```

a anotação customizada possui um atributo do tipo String[] que recebe as String dos tipos de MediaTypes permitidos.

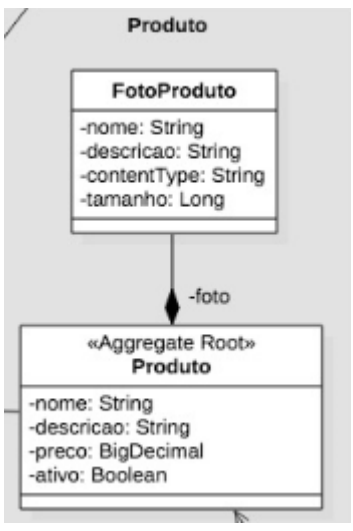
```
package com.algaworks.algafood.core.validation;

import ...

4 usages
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE, ElementType.CONSTRUCTOR,
        ElementType.PARAMETER, ElementType.TYPE_USE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {FileContentTypeValidator.class})
public @interface FileContentType {
    String message() default "A foto dever ser do tipo JPG ou PNG";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    2 usages
    String[] allowed();
}
```

```
package com.algaworks.algafood.core.validation;

import ...

1 usage
public class FileContentTypeValidator implements ConstraintValidator<FileContentType, MultipartFile> {

    2 usages
    String[] allowed;

    @Override
    public void initialize(FileContentType constraintAnnotation) {
        this.allowed = constraintAnnotation.allowed();
    }

    @Override
    public boolean isValid(MultipartFile file,
                           ConstraintValidatorContext constraintValidatorContext) {

        ArrayList<String> allowedContentTypes = new ArrayList<>(Arrays.asList(allowed));

        return allowedContentTypes.contains(file.getContentType());
    }
}
```

```
#14.4. Desafio: Validando o content type do arquivo
FileContentType.fotoProdutoInput.file=A foto deve ser do tipo JPG ou PNG
```

14.5. Mapeando entidade FotoProduto e relacionamento um-para-um

domingo, 2 de abril de 2023 15:11



Há um mapeamento OneToOne

```java
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class FotoProduto {

    @EqualsAndHashCode.Include
    @Id
    @Column(name = "produto_id")
    private Long id;

    @OneToOne(fetch = FetchType.LAZY)
    @MapsId
    private Produto produto;

    private String nomeArquivo;
    private String descricao;
    private String contenttype;
    private Long tamanho;

}
```

https://blog.algaworks.com/lazy-loading-com-mapeamento-onetoone/

14.6. Implementando serviço de cadastro de foto de produto

domingo, 2 de abril de 2023 16:33

```java
package com.algaworks.algafood.domain.repository;

import com.algaworks.algafood.domain.model.FotoProduto;

3 usages  2 implementations
public interface ProdutoRepositoryQueries {

    1 implementation
    FotoProduto save(FotoProduto fotoProduto);
}
```

```java
@Repository
public class ProdutoRepositoryImpl implements ProdutoRepositoryQueries {

    1 usage
    @PersistenceContext
    private EntityManager manager;

    @Transactional
    @Override
    public FotoProduto save(FotoProduto fotoProduto) {
        return manager.merge(fotoProduto);
    }
}
```

```java
@Repository
public interface ProdutoRepository extends CustomJpaRepository<Produto, Long>, ProdutoRepositoryQueries{

    1 usage
    @Query("from Produto where id = :produtoId and restaurante.id = :restauranteId")
    Optional<Produto> findByProdutoAndRestaurante(Long produtoId, Long restauranteId);

    1 usage
    List<Produto> findByRestaurante(Restaurante restaurante);

    1 usage
    @Query("from Produto p where p.ativo = true and p.restaurante = :restaurante")
    List<Produto> findByAtivosRestaurante(Restaurante restaurante);
}
```

```java
@PutMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public FotoProdutoDTO atualizarFoto(@PathVariable Long restauranteId,
                                    @PathVariable Long produtoId,
                                    @Valid FotoProdutoInput fotoProdutoInput) {

    FotoProduto fotoProduto = new FotoProduto();
    final MultipartFile file = fotoProdutoInput.getFile();

    fotoProduto.setProduto(produtoService.buscar(produtoId, restauranteId));
    fotoProduto.setDescricao(fotoProdutoInput.getDescricao());
    fotoProduto.setContentType(file.getContentType());
    fotoProduto.setTamanho(file.getSize());
    fotoProduto.setNomeArquivo(file.getOriginalFilename());

    final FotoProduto fotoSalva = catalogoFotoProdutoService.salvar(fotoProduto);

    return fTAssember.toDTO(fotoSalva);
}
```

```java
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class FotoProduto {

    @EqualsAndHashCode.Include
    @Id
    @Column(name = "produto_id")
    private Long id;

    @OneToOne(fetch = FetchType.LAZY)
    @MapsId
    private Produto produto;

    private String nomeArquivo;
    private String descricao;
    private String contentType;
    private Long tamanho;

}
```

```java
2 usages
@Getter
@Setter
public class FotoProdutoInput {

    @NotNull
    @FileSize(max = "500KB")
    @FileContentType(allowed = {MediaType.IMAGE_JPEG_VALUE, MediaType.IMAGE_PNG_VALUE})
    private MultipartFile file;

    @NotBlank
    private String descricao;

}
```

Nota: Ocorre erro ao tentar enviar uma foto novamente para um mesmo id de produto

14.7. Excluindo e substituindo cadastro de foto de produto

domingo, 2 de abril de 2023 17:56

Como foi implementado o serviço de cadastro de foto de produto, ao tentar novamente uma foto para um produto, acusava erro de violação de constraint, pois estava inserindo um registro com id duplicado, no caso, id do produto. Para alterar o comportamento, a lógica seguida foi, ao salvar uma foto que já exista associada ao produto, o registro atual no banco de dados será deletado para a nova foto ser inserida.

```java
@Transactional
public FotoProduto salvar(FotoProduto fotoProduto){
    Long restauranteId = fotoProduto.getRestauranteId();
    final Long produtoId = fotoProduto.getProduto().getId();

    final Optional<FotoProduto> fotoExistente = produtoRepository.
            findFotoById(restauranteId, produtoId);

    fotoExistente.ifPresent(produto -> produtoRepository.delete(produto));

    return produtoRepository.save(fotoProduto);
}
```

- Verificar se a foto existe no banco de dados baseado no id do restaurante e do produto associados
- Caso for presente, o método ifPresent implementado por uma expressão lambda de para um Consumer vai ser chamado, deletando a entidade(registro) para salvar a nova foto

- Implementar Query para encontrar uma foto

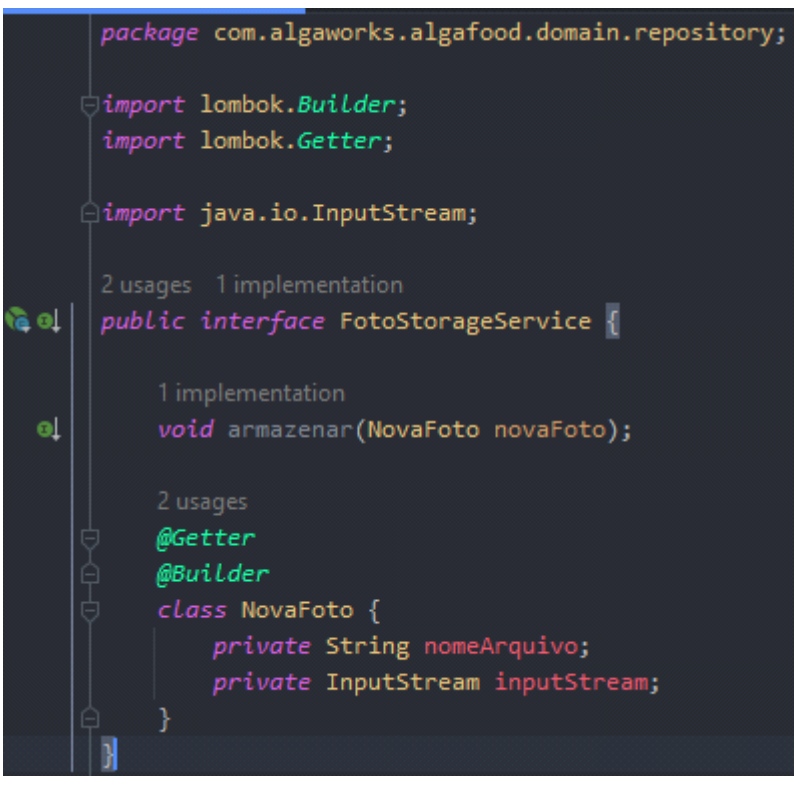
```java
1 usage
@Query("select f from FotoProduto f join f.produto p where p.restaurante.id = :restauranteId and p.id = :produtoId")
Optional<FotoProduto> findFotoById(Long restauranteId, Long produtoId);
```

```java
3 usages  2 implementations
public interface ProdutoRepositoryQueries {

    1 implementation
    FotoProduto save(FotoProduto fotoProduto);
    1 usage  1 implementation
    void delete(FotoProduto fotoProduto);
}
```

14.8. Implementando o serviço de armazenagem de fotos no disco local

domingo, 2 de abril de 2023 19:13

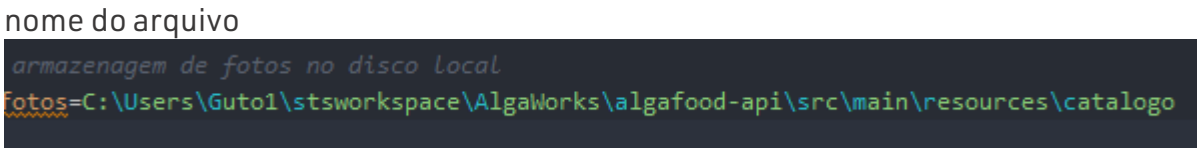
- Criar interface que provê métodos de armazenamento



O método recebe um arquivo que representa uma foto, o não uso da classe MultipartFile é devido ao domínio que a classe representa, voltada para web, e a implementação atual diz respeito a um nível de armazenamento local. A classe NovaFoto guarda o fluxo de bytes do arquivo e o nome do mesmo para ser persistido localmente.

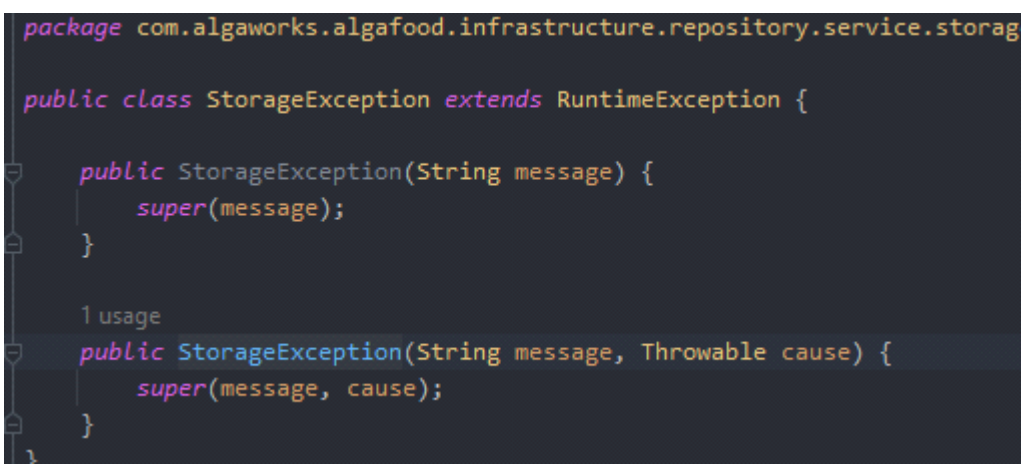
- Criar implementação da interface

O método getArquivoPath foi criado para gerar um novo Path (diretório) junto ao nome do arquivo



ficando: "...\resources\catalogo\foto2.jpg"

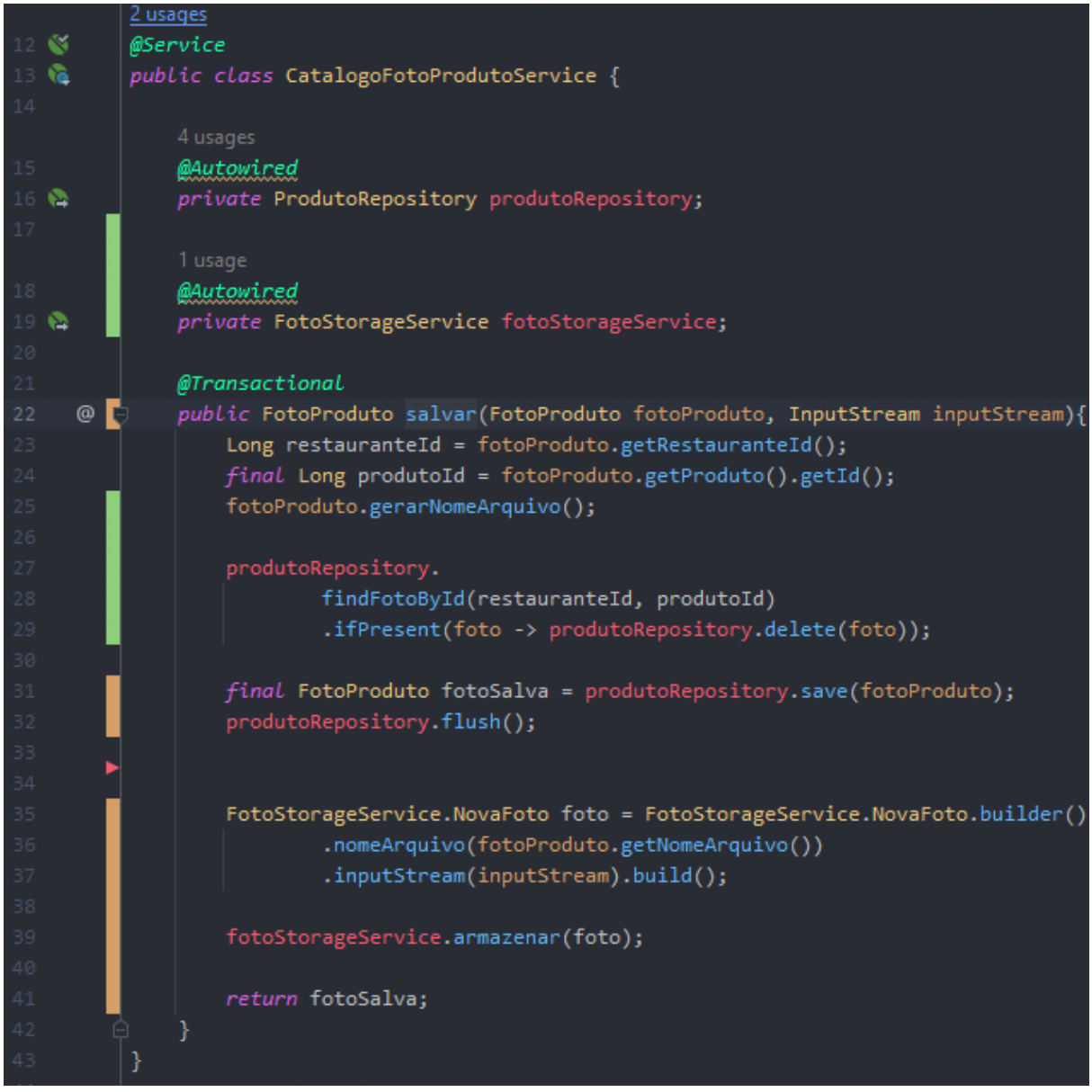
FileCopyUtils.copy recebe no 1º argumento um fluxo de entrada do arquivo e o 2º argumento recebe o fluxo de saída junto ao path para a persistência local



14.9. Integrando o serviço de catálogo de fotos com o serviço de armazenagem

domingo, 2 de abril de 2023 20:12

- A camada de serviço foi criada na aula 14.8 e nesta aula ocorre a integralização do serviço de armazenagem local e com o serviço do banco de dados



```
2 usages
@Service
public class CatalogoFotoProdutoService {

    4 usages
    @Autowired
    private ProdutoRepository produtoRepository;

    1 usage
    @Autowired
    private FotoStorageService fotoStorageService;

    @Transactional
    public FotoProduto salvar(FotoProduto fotoProduto, InputStream inputStream){
        Long restauranteId = fotoProduto.getRestauranteId();
        final Long produtoId = fotoProduto.getProduto().getId();
        fotoProduto.gerarNomeArquivo();

        produtoRepository.
                findFotoById(restauranteId, produtoId)
                .ifPresent(foto -> produtoRepository.delete(foto));

        final FotoProduto fotoSalva = produtoRepository.save(fotoProduto);
        produtoRepository.flush();


        FotoStorageService.NovaFoto foto = FotoStorageService.NovaFoto.builder()
                .nomeArquivo(fotoProduto.getNomeArquivo())
                .inputStream(inputStream).build();

        fotoStorageService.armazenar(foto);

        return fotoSalva;
    }
}
```

o controlador chama o serviço de foto de produtos passando um objeto do tipo FotoProduto

```java
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class FotoProduto {

    @EqualsAndHashCode.Include
    @Id
    @Column(name = "produto_id")
    private Long id;

    2 usages
    @OneToOne(fetch = FetchType.LAZY)
    @MapsId
    private Produto produto;

    2 usages
    private String nomeArquivo;
    private String descricao;
    private String contentType;
    private Long tamanho;

    public Long getRestauranteId(){
        if(this.produto != null){
            return this.produto.getRestaurante().getId();
        }
        return null;
    }

    1 usage
    public void gerarNomeArquivo(){
        this.nomeArquivo = UUID.randomUUID().toString() + "_" + this.nomeArquivo;
    }
}
```

que representa um arquivo da foto contendo a foto e suas informações vindos da requisição, o objeto FotoProduto é instanciado e gerido através de um arquivo MultipartFile

```java
@Getter
@Setter
public class FotoProdutoInput {

    @NotNull
    @FileSize(max = "500KB")
    @FileContentType(allowed = {MediaType.IMAGE_JPEG_VALUE, MediaType.IMAGE_PNG_VALUE})
    private MultipartFile file;

    @NotBlank
    private String descricao;

}
```

o 2º argumento de CatalogoFotoProdutoService.salvar(FotoProduto,InputStream) é um inputStream gerido também do objeto MultipartFile vindo da requisição.

```java
@Transactional
public FotoProduto salvar(FotoProduto fotoProduto, InputStream inputStream){
    Long restauranteId = fotoProduto.getRestauranteId();
    final Long produtoId = fotoProduto.getProduto().getId();
    fotoProduto.gerarNomeArquivo();

    produtoRepository.
            findFotoById(restauranteId, produtoId)
            .ifPresent(foto -> produtoRepository.delete(foto));

    final FotoProduto fotoSalva = produtoRepository.save(fotoProduto);
    produtoRepository.flush();


    FotoStorageService.NovaFoto foto = FotoStorageService.NovaFoto.builder()
            .nomeArquivo(fotoProduto.getNomeArquivo())
            .inputStream(inputStream).build();

    fotoStorageService.armazenar(foto);

    return fotoSalva;
}
```

o método gerarNomeArquivo dentro do objeto FotoProduto altera o nome original do arquivo adicionando um UUID + "_" + nome original, para evitar conflitos de nomes no storage local.

o método da linha 27 procura uma foto baseado no produtoId e o restauranteId associados e deleta caso já contenha no banco de dados, para evitar o erro de salvar um id duplicado, visto na aula 14.7

```java
@Override
public void armazenar(NovaFoto novaFoto) {

    try {
        Path arquivoPath = getArquivoPath(novaFoto.getNomeArquivo());

        FileCopyUtils.copy(novaFoto.getInputStream(), Files.newOutputStream(arquivoPath));
    } catch (Exception e) {
        throw new StorageException("Não foi possível armazenar arquivo.", e);
    }
}

1 usage
private Path getArquivoPath(String nomeArquivo) { return path.resolve(Path.of(nomeArquivo)); }
}
```

14.10. Implementando a remoção e substituição de arquivos de fotos no serviço de armazenagem

domingo, 2 de abril de 2023 23:22

Ao substituir um cadastro de foto do produto, a antiga foto do storage local não é removida, ocasionando em lixo no local.

- Criar método na interface FotoStorageService

```
public interface FotoStorageService {

    1 usage   1 implementation
    void armazenar(NovaFoto novaFoto);

    1 usage   1 implementation
    void remover(String nomeArquivo);

    4 usages
    @Getter
    @Builder
    class NovaFoto {
        private String nomeArquivo;
        private InputStream inputStream;
    }
}
```

- Implementar métodos na classe LocalFotoStorageService

```
@Service
public class LocalFotoStorageService implements FotoStorageService {

    1 usage
    @Value("${algafood.storage.local.diretorio-fotos}")
    private Path path;

    1 usage
    @Override
    public void armazenar(NovaFoto novaFoto) {...}

    1 usage
    @Override
    public void remover(String nomeArquivo) {
        try {
            final Path arquivoPath = getArquivoPath(nomeArquivo);
            Files.deleteIfExists(arquivoPath);
        } catch (IOException e) {
            throw new StorageException("Não foi possível deletar arquivo.", e);
        }
    }

    2 usages
    private Path getArquivoPath(String nomeArquivo) {
        return path.resolve(Path.of(nomeArquivo));
    }
}
```

A variável path contém o diretório da pasta onde vai ser armazenada e o método getArquivoPath concatena o diretorio e o nome do arquivo, para criar um novo Path contendo o arquivo com seu caminho para finalizar o delete.

- O método para deletar o arquivo no storage local só vai ser chamado caso existir a foto no banco de dados, e logo, também vai estar no storage local

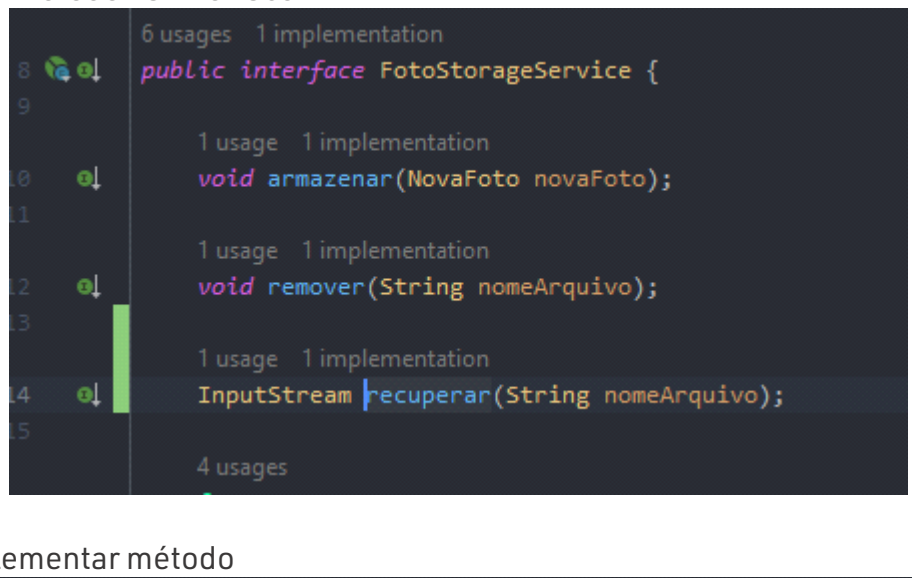
```
@Transactional
public FotoProduto salvar(FotoProduto fotoProduto, InputStream inputStream){
    Long restauranteId = fotoProduto.getRestauranteId();
    final Long produtoId = fotoProduto.getProduto().getId();
    fotoProduto.gerarNomeArquivo();


    produtoRepository.
            findFotoById(restauranteId, produtoId)
            .ifPresent(foto -> {
                fotoStorageService.remover(foto.getNomeArquivo());
                produtoRepository.delete(foto);
            });

    final FotoProduto fotoSalva = produtoRepository.save(fotoProduto);
```

Se a foto do produto, baseado no id do produto e no restaurante onde esse produto está associado, já existir no banco de dados, o mesmo vai ser deletado e também deletado do local storage.

14.11. Desafio: Implementando recuperação de foto no serviço de armazenagem

segunda-feira, 3 de abril de 2023 08:51

- Criar método na interface



- Implementar método

```java
@Service
public class LocalFotoStorageService implements FotoStorageService {

    1 usage
    @Value("${algafood.storage.local.diretorio-fotos}")
    private Path path;

    1 usage
    @Override
    public void armazenar(NovaFoto novaFoto) {...}

    1 usage
    @Override
    public void remover(String nomeArquivo) {...}

    1 usage
    @Override
    public InputStream recuperar(String nomeArquivo) {
        InputStream aquivoRecuperado;
        try {
            final Path arquivoPath = getArquivoPath(nomeArquivo);

            aquivoRecuperado = Files.newInputStream(arquivoPath);
        } catch (Exception e) {
            throw new StorageException("Não foi possível recuperar arquivo.", e);
        }

        return aquivoRecuperado;
    }

    3 usages
    private Path getArquivoPath(String nomeArquivo) {
        return path.resolve(Path.of(nomeArquivo));
    }
}
```

14.12. Desafio: Implementando endpoint de consulta de foto de produto

segunda-feira, 3 de abril de 2023 11:15

- Implementar endpoint
  - GET restaurantes/{restauranteId}/produtos/{produtoId}/foto

```java
@GetMapping
public FotoProdutoDTO buscar(@PathVariable Long restauranteId,
                             @PathVariable Long produtoId) {
    FotoProduto fotoProduto = catalogoFotoProdutoService.buscarOuFalhar(restauranteId, produtoId);
    final FotoProdutoDTO fotoProdutoDTO = fTAssember.toDTO(fotoProduto);

    return fotoProdutoDTO;
}
```

14.13. Servindo arquivos de fotos pela API

segunda-feira, 3 de abril de 2023 11:56

- Implementar no controlador um método que retorna um tipo de InputStream da foto buscada no local storage
- Utilizar Content Negotiation na busca de uma foto

☑	Accept	image/jpeg
☐	Accept	application/json

```java
@GetMapping(produces = {MediaType.IMAGE_JPEG_VALUE})
public ResponseEntity<InputStreamResource> servirFoto(@PathVariable Long restauranteId,
                                                      @PathVariable Long produtoId){
    try{
        FotoProduto fotoProduto = catalogoFotoProdutoService.buscarOuFalhar(restauranteId, produtoId);

        final InputStream inputStream = fotoStorageService.recuperar(fotoProduto.getNomeArquivo());

        return ResponseEntity
                .ok()
                .contentType(MediaType.IMAGE_JPEG)
                .body(new InputStreamResource(inputStream));
    } catch(EntidadeNaoEncontradaException e){
        return ResponseEntity.notFound().build();
    }
}
```

```java
26 usages
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class FotoProduto {

    @EqualsAndHashCode.Include
    @Id
    @Column(name = "produto_id")
    private Long id;

    2 usages
    @OneToOne(fetch = FetchType.LAZY)
    @MapsId
    private Produto produto;

    2 usages
    private String nomeArquivo;
    private String descricao;
    private String contentType;
    private Long tamanho;
```

```java
@Override
public InputStream recuperar(String nomeArquivo) {
    InputStream aquivoRecuperado;
    try {
        final Path arquivoPath = getArquivoPath(nomeArquivo);

        aquivoRecuperado = Files.newInputStream(arquivoPath);
    } catch (Exception e) {
        throw new StorageException("Não foi possível recuperar arquivo.", e);
    }

    return aquivoRecuperado;
}
```

GET http://localhost:8080/restaurantes/1/produtos/1/foto

Params Authorization Headers (10) Body Pre-request Script Tests Settings

☑	Content-Length	<calculated when request is sent>
☑	Host	<calculated when request is sent>
☑	User-Agent	PostmanRuntime/7.31.3
☑	Accept	*/*
☑	Accept-Encoding	gzip, deflate, br
☑	Connection	keep-alive
☑	Accept	image/jpeg
☐	Accept	application/json
	Key	Value

Body Cookies Headers (5) Test Results

A surpresa no rosto quando você mostra para alguém que o brócolis não ocorreu "naturalmente". Humanos o criaram a partir seleção da mostarda silvestre

Brassica oleracea

```java
56
57      @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
58      public FotoProdutoDTO buscar(@PathVariable Long restauranteId,
59                                   @PathVariable Long produtoId) {...}
65
66      @GetMapping(produces = {MediaType.IMAGE_JPEG_VALUE})
67      public ResponseEntity<InputStreamResource> servirFoto(@PathVariable Long restauranteId,
68                                                            @PathVariable Long produtoId){...}
82
83  }
```

14.14. Checando media type ao servir arquivos de fotos

segunda-feira, 3 de abril de 2023 12:42

- Atualmente, o endpoint só aceita na resposta o tipo image/JPEG


```
@GetMapping(produces = {MediaType.IMAGE_JPEG_VALUE})
public ResponseEntity<InputStreamResource> servirFoto(@PathVariable Long restauranteId,
```

```
return ResponseEntity
        .ok()
        .contentType(MediaType.IMAGE_JPEG)
        .body(new InputStreamResource(inputStream));
```

- Implementar método do controlador para tratar diferentes MediaTypes da requisição

```
@GetMapping
public ResponseEntity<InputStreamResource> servirFoto(@PathVariable Long restauranteId,
                                                      @PathVariable Long produtoId,
                                                      @RequestHeader(name = "accept") String acceptHeader) throws HttpMediaTypeNotAcceptableException {

        FotoProduto fotoProduto = catalogoFotoProdutoService.buscarOuFalhar(restauranteId, produtoId);
        MediaType mediaTypeFotoProduto = MediaType.parseMediaType(fotoProduto.getContentType());
        List<MediaType> mediaTypesAceitas = MediaType.parseMediaTypes(acceptHeader);

        verificarCompatibilidadeMediaType(mediaTypeFotoProduto, mediaTypesAceitas);

        try{
        final InputStream inputStream = fotoStorageService.recuperar(fotoProduto.getNomeArquivo());

        return ResponseEntity
                .ok()
                .contentType(mediaTypeFotoProduto)
                .body(new InputStreamResource(inputStream));
    } catch(EntidadeNaoEncontradaException e){
        return ResponseEntity.notFound().build();
    }
}
```

```
1 usage
private void verificarCompatibilidadeMediaType(MediaType mediaType,
                                               List<MediaType> mediaTypesAceitas)
        throws HttpMediaTypeNotAcceptableException {
    /*false para não compativel com algum mediatype da lista*/
    final boolean compativel = mediaTypesAceitas.stream().anyMatch(m -> m.isCompatibleWith(mediaType));

    if(!compativel){/*false vira true e true vira fase*/
        throw new HttpMediaTypeNotAcceptableException(mediaTypesAceitas);
    }
}
```

14.15. Desafio: implementando endpoint de exclusão de foto de produto

terça-feira, 4 de abril de 2023 09:15

- DELETE restaurantes/{restauranteId}/produtos/{produtoId}/foto
  - Criar método no controlador

```java
@DeleteMapping
public void deletarFotoProduto(@PathVariable Long restauranteId,
                               @PathVariable Long produtoId){

    catalogoFotoProdutoService.deletar(restauranteId, produtoId);

}
```

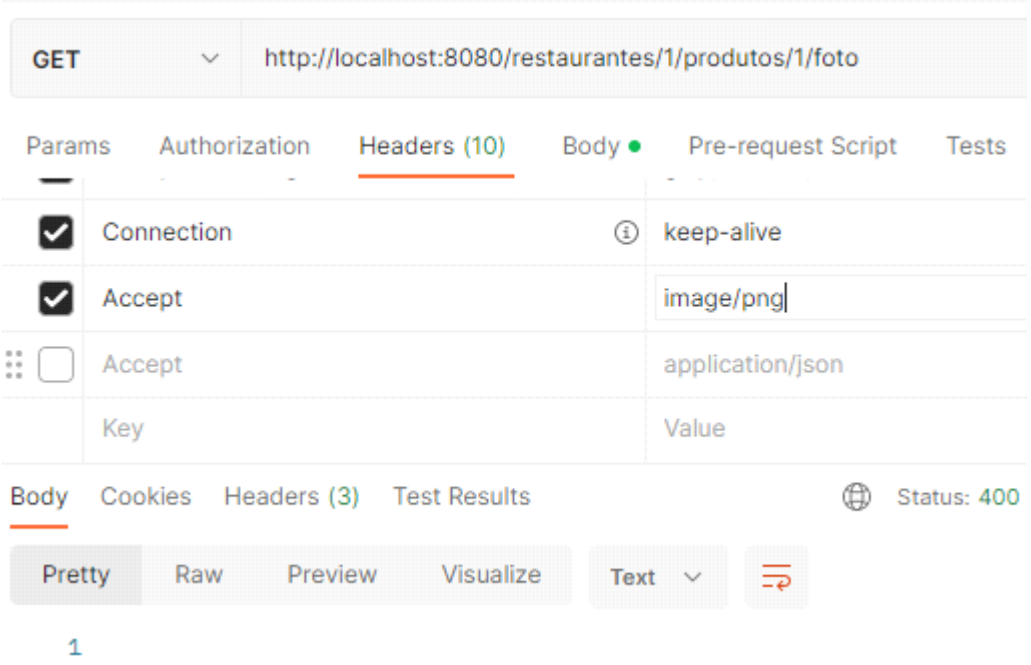
- Criar método no serviço de fotos de produtos

```java
@Transactional
public void deletar(Long restauranteId, Long produtoId) {
    FotoProduto fotoProduto = buscarOuFalhar(restauranteId, produtoId);
    fotoStorageService.remover(fotoProduto.getNomeArquivo());
    produtoRepository.delete(fotoProduto);

    produtoRepository.flush();
    fotoStorageService.remover(fotoProduto.getNomeArquivo());
}
```

14.16. Corrigindo exception handler de media type não aceita

terça-feira, 4 de abril de 2023 10:02

- Ao fazer uma requisição especificando no header o 'accept' e como uma imagem ou outro formato que não suporta serialização



o erro será tratado, porém, como estamos utilizando a RFC problem details, a representação do erro não poderá serializar para JSON, pois o header diz que está aceitando apenas o tipo de media especificada.

```
@Override
protected ResponseEntity<Object> handleHttpMediaTypeNotAcceptable(HttpMediaTypeNotAcceptableException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {
    return ResponseEntity.status(status).headers(headers).build();
}
```

14.17. Amazon S3: conhecendo o serviço de storage da AWS

terça-feira, 4 de abril de 2023 14:02

- Criar conta gratuita na Amazon
- Criar novo bucket
- Propriedade do objeto
    - Selecione a opção ACLs habilidades, assim como a opção Autor do objeto.
- Desabilite todas opções, com exceção da opção "Desativar o bloqueio de todo o acesso público...".
- As demais opções como Versionamento de bucket, Criptografia padrão e Configurações avançadas, podem permanecer desativadas.
- CORS
    - Na tela o bucket, clique na aba Permissões, procure pela opção Compartilhamento de recursos de origem cruzada (CORS). Em seguida clique em editar.

```
[
  {
    "AllowedHeaders": [],
    "AllowedMethods": [
      "PUT",
      "POST",
      "DELETE",
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

14.18. Criando usuário com permissões para adicionar objetos na Amazon S3

terça-feira, 4 de abril de 2023 14:07

14.19. Criando chaves de acesso para a API da AWS

terça-feira, 4 de abril de 2023 14:57

14.20. Criando bean de propriedades de configuração dos serviços de storage

terça-feira, 4 de abril de 2023 14:59

- Criando configurações de acesso a api da Amazon S3
  - Definindo as credenciais de acesso ao S3 via application.properties

```
#14.20. Criando bean de propriedades de configuração dos serviços de storage
#algafood.storage.s3.id-chave-acesso=environment variables in intelliJ
#algafood.storage.s3.chave-acesso-secreta=environment variables in intelliJ
algafood.storage.s3.bucket=algaworks-algafood-wgustavosantos
algafood.storage.s3.regiao=us-east-1
algafood.storage.s3.diretorio-fotos=catalogo
```

```
#14.8. Implementando o serviço de armazenagem de fotos no disco local
algafood.storage.local.diretorio-fotos=C:\\Users\\Guto1\\Documents\\catalogo
```

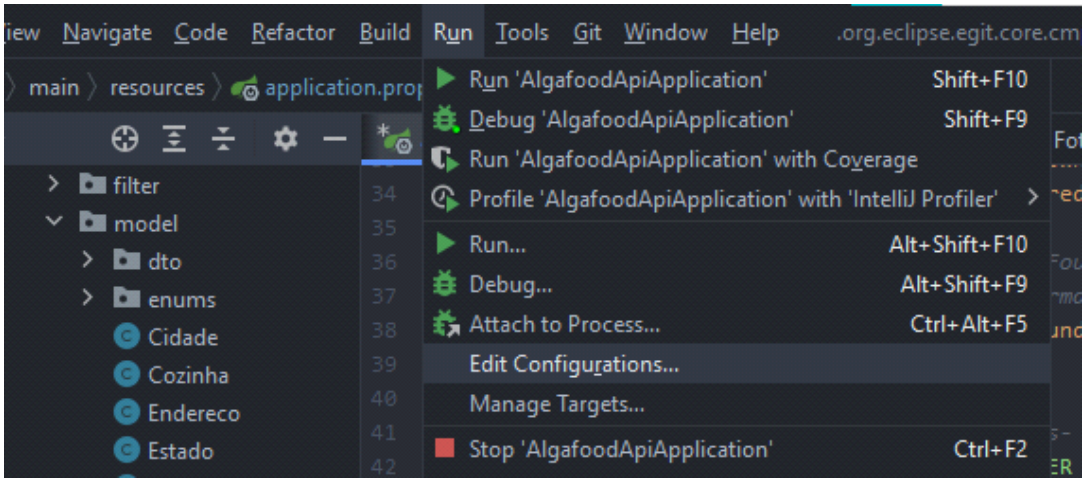
- Alternativa: criar classe de configurações de propriedades do application.properties

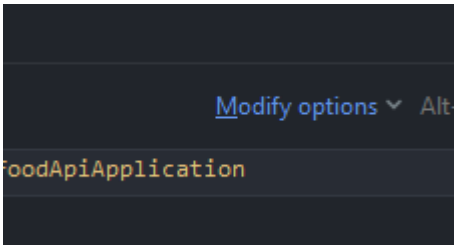
```
package com.algaworks.algafood.core.storage;

import ...

2 usages
@Getter
@Setter
@Component
@ConfigurationProperties("algafood.storage")
public class StorageProperties {

    private Local local = new Local();
    private S3 s3 = new S3();

    @{...}
    public class Local {
        private Path diretorioFotos;
    }

    2 usages
    @Getter
    @Setter
    public class S3{
        private String idChaveAcesso;
        private String chaveAcessoSecreta;
        private String bucket;
        private String regiao;
        private String diretorioFotos;
    }
}
```

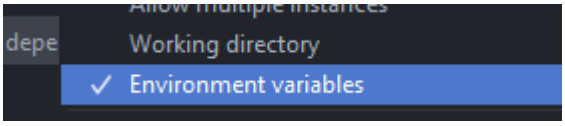
- @configurationProperties: definir as propriedades hierárquicas baseadas pelo nome, o valor padrão é o prefixo. O Spring é inteligente o suficiente para mapear os caminhos das propriedades para objetos alinhados. a classe

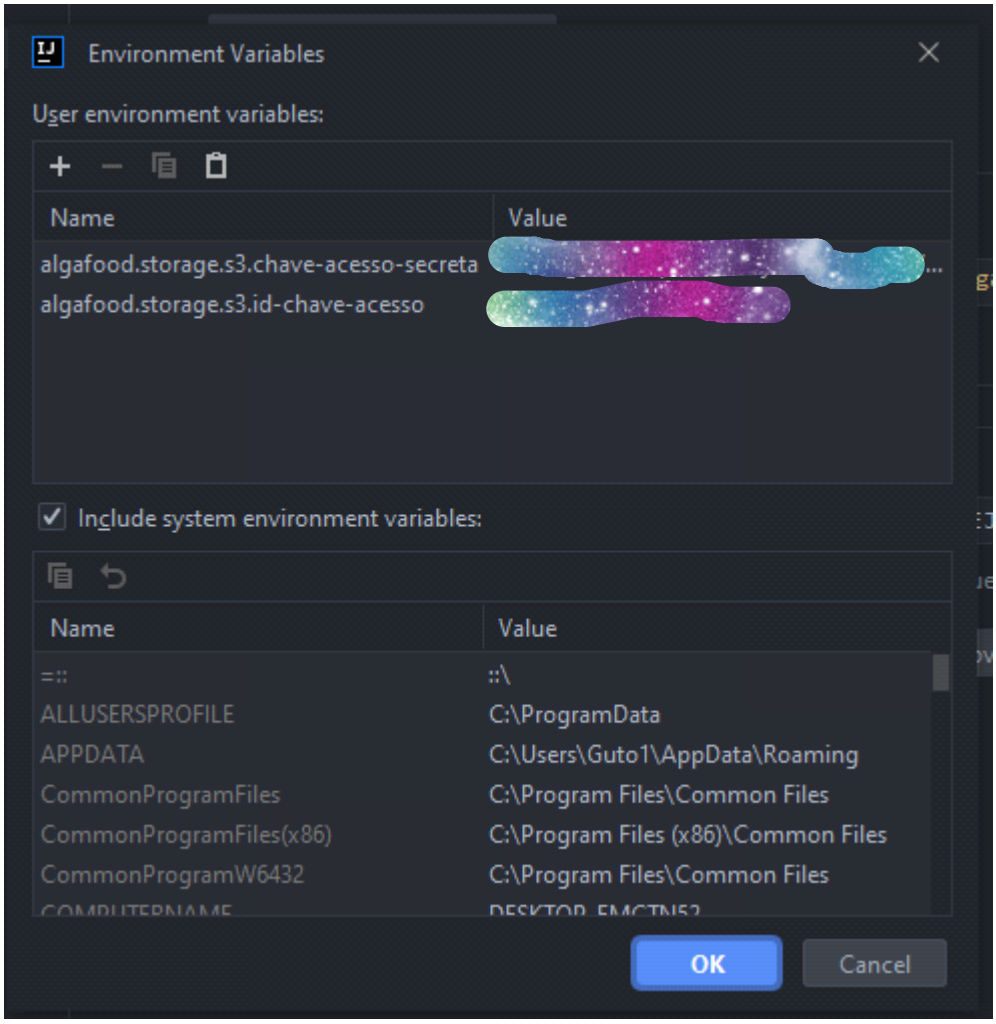
StorageProperties representa o prefixo "algafood.storage" e suas classes internas representam .local e .s3 respectivamente. suas propriedades representam .diretorio-local e .id-chave-acesso, .chave-acesso-secreta, .bucket, .regiao e .diretorio-fotos respectivamente.

dados sensíveis no application.properties podem ser adicionadas via variáveis de ambiente.








Environment Variables
User environment variables:
Name
Value
algafood.storage.s3.chave-acesso-secreta
algafood.storage.s3.id-chave-acesso
Include system environment variables:
Name
Value
=::
::\
ALLUSERSPROFILE
C:\ProgramData
APPDATA
C:\Users\Guto1\AppData\Roaming
CommonProgramFiles
C:\Program Files\Common Files
CommonProgramFiles(x86)
C:\Program Files (x86)\Common Files
CommonProgramW6432
C:\Program Files\Common Files
OK
Cancel

14.21. Adicionando o SDK Java da Amazon S3 no projeto e criando classe do serviço de storage

terça-feira, 4 de abril de 2023 15:56

- Utilizar o SDK da amazon para interagir com o serviço S3
  <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-s3 -->
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-s3</artifactId>
    <version>1.12.441</version>
  </dependency>

- Criar classe de implementação

```java
package com.algaworks.algafood.infrastructure.service.storage;

import com.algaworks.algafood.domain.repository.FotoStorageService;
import org.springframework.stereotype.Service;

import java.io.InputStream;

@Service
public class S3FotoStorageService implements FotoStorageService {
    1 usage
    @Override
    public void armazenar(NovaFoto novaFoto) {

    }

    3 usages
    @Override
    public void remover(String nomeArquivo) {

    }

    2 usages
    @Override
    public InputStream recuperar(String nomeArquivo) {
        return null;
    }
}
```

14.22. Definindo bean do client da Amazon S3 e configurando credenciais

terça-feira, 4 de abril de 2023 16:14

- A partir do SDK da Amazon s3 configurado no pom.xml na aula 14.2, temos acesso ao conjunto de biblioteca da Amazon S3.
- Injetar instância da interface Amazons3

```java
package com.algaworks.algafood.infrastructure.service.storage;

import ...

@Service
public class S3FotoStorageService implements FotoStorageService {

    @Autowired
    private AmazonS3 amazonS3;
    1 usage
    @Override
    public void armazenar(NovaFoto novaFoto) {

    }

    3 usages
    @Override
    public void remover(String nomeArquivo) {

    }
```

- Configurar Bean para a interface

```java
package com.algaworks.algafood.core.storage;

import ...

@Configuration
public class AmazonS3Config {

    3 usages
    @Autowired
    private StorageProperties storageProperties;

    @Bean
    public AmazonS3 amazonS3(){

        final String idChaveAcesso = storageProperties.getS3().getIdChaveAcesso();
        final String chaveAcessoSecreta = storageProperties.getS3().getChaveAcessoSecreta();
        Regions regiao = storageProperties.getS3().getRegiao();

        final BasicAWSCredentials basicAWSCredentials = new BasicAWSCredentials(idChaveAcesso, chaveAcessoSecreta);

        return AmazonS3ClientBuilder.standard()
                .withCredentials(new AWSStaticCredentialsProvider(basicAWSCredentials)).
                withRegion(regiao).
                build();
    }
}
```

- Refatorar atributo da classe StorageProperties.S3

```java
package com.algaworks.algafood.core.storage;

import ...

3 usages
@Getter
@Setter
@Component
@ConfigurationProperties("algafood.storage")
public class StorageProperties {

    private Local local = new Local();
    private S3 s3 = new S3();

    2 usages
    @Getter
    @Setter
    public class Local {
        private Path diretorioFotos;
    }

    2 usages
    @Getter
    @Setter
    public class S3{
        private String idChaveAcesso;
        private String chaveAcessoSecreta;
        private String bucket;
        private Regions regiao;
        private String diretorioFotos;
    }
}
```

- a classe interna String regiao foi substituída para Regions do pacote com.amazonaws.regions, é uma Enumeração e por isso o Spring consegue atribuir valores facilmente a partir do application.properties

```
#14.8. Implementando o serviço de armazenagem de fotos no disco local
algafood.storage.local.diretorio-fotos=C:\\Users\\Guto1\\Documents\\catalogo

#14.20. Criando bean de propriedades de configuração dos serviços de storage
#algafood.storage.s3.id-chave-acesso=environment variables in intelliJ
#algafood.storage.s3.chave-acesso-secreta=environment variables in intelliJ
algafood.storage.s3.bucket=algaworks-algafood-wgustavosantos
algafood.storage.s3.regiao=us_east_1
algafood.storage.s3.c
    us_east_1
    us_gov_east_1
    us_iso_east_1
    us_isob_east_1
```

-

Nota: adicionar no pom.xml o processador de anotação do Spring

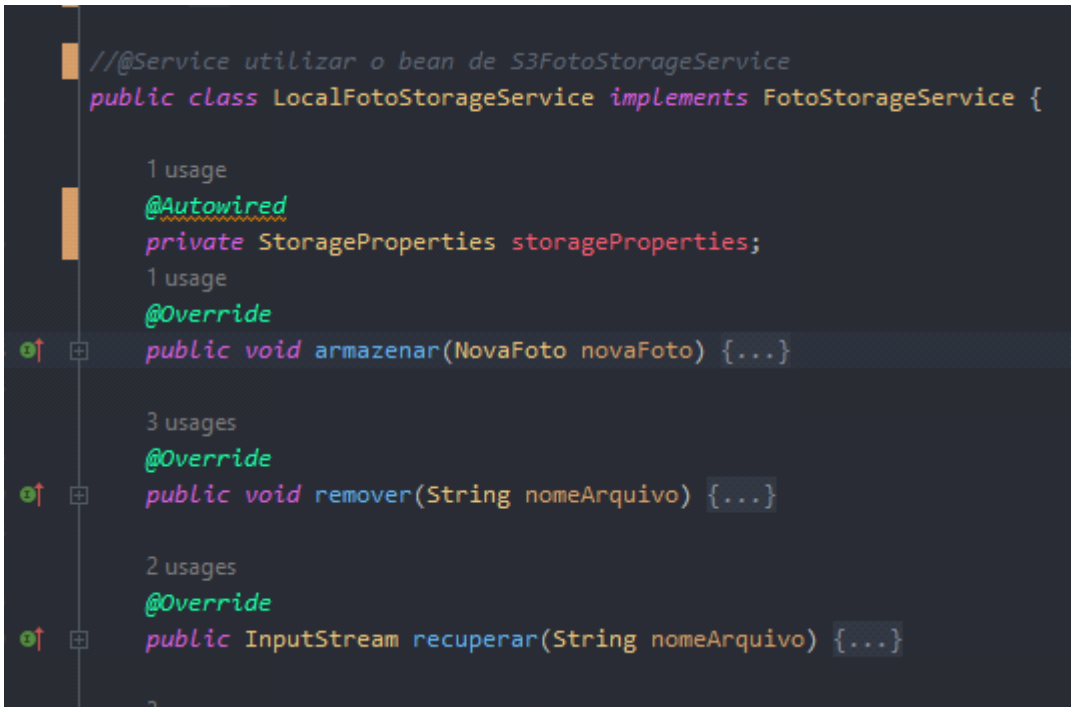
```xml
<!--é um processador de anotação que gera metadados sobre classes em seu aplicativo que são anotadas com
 @ConfigurationProperties. Esses metadados são usados pelo seu IDE (Eclipse, IntelliJ ou NetBeans)
 para fornecer preenchimento automático e documentação para as propriedades ao editar application.properties-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

14.23. Implementando a inclusão de objetos no bucket da Amazon S3

terça-feira, 4 de abril de 2023 16:44

Tempos 1 interface para duas implementações, uma para disco local e outra para armazenar na nuvem com amazon S3. A LocalFotoStorageService não está sendo gerenciada pelo Spring para o Bean de S3 ser injetado.

FotoStorageService implementada por LocalFotoStorageService



FotoStorageService implementada por S3FotoStorageService

```java
@Service
public class S3FotoStorageService implements FotoStorageService {

    1 usage
    @Autowired
    private AmazonS3 amazonS3;

    2 usages
    @Autowired
    private StorageProperties storageProperties;

    1 usage
    @Override
    public void armazenar(NovaFoto novaFoto) {...}

    1 usage
    private String getCaminhoArquivo(String nomeArquivo) {...}
```

A classe de serviço CatalogoFotoProdutoService está usando os beans das implementações, que é chamado pelo método controlador.

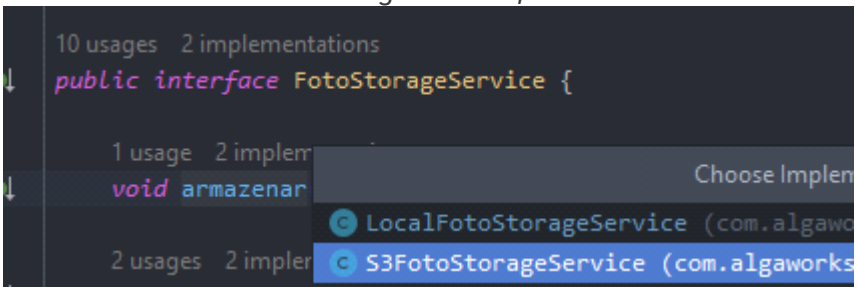
```java
S3FotoStorageService.java  CatalogoFotoProdutoService.java

10
11  import java.io.InputStream;
12
    2 usages
13  @Service
14  public class CatalogoFotoProdutoService {
15
        6 usages
16      @Autowired
17      private ProdutoRepository produtoRepository;
18
        5 usages
19      @Autowired
20      private FotoStorageService fotoStorageService;
21
22      @Transactional
```

A classe CatalogoProdutoService está gerenciando a 'foto do produto' para persistir na nuvem e no banco de dados, a classe integra o armazenamento tanto no banco quanto na nuvem. Como no método Salvar:

```java
@Transactional
public FotoProduto salvar(FotoProduto fotoProduto, InputStream inputStream) {
    Long restauranteId = fotoProduto.getRestauranteId();
    final Long produtoId = fotoProduto.getProduto().getId();
    fotoProduto.gerarNomeArquivo();


    produtoRepository.
            findFotoById(restauranteId, produtoId)
            .ifPresent(foto -> {
                fotoStorageService.remover(foto.getNomeArquivo());
                produtoRepository.delete(foto);
                produtoRepository.delete(foto);
            });

    final FotoProduto fotoSalva = produtoRepository.save(fotoProduto);


    FotoStorageService.NovaFoto foto = FotoStorageService.NovaFoto.builder()
            .nomeArquivo(fotoProduto.getNomeArquivo())
            .inputStream(inputStream).build();

    fotoStorageService.armazenar(foto);

    return fotoSalva;
}
```

Para armazenar a foto na nuvem:

o método fotoStorageService.armazenar(foto) da classe CatalogoFotoProdutoService chama o Bean da interface FotoStorageService, Bean esse da classe S3FotoStorage




```java
@Service
public class S3FotoStorageService implements FotoStorageService {

    1 usage
    @Autowired
    private AmazonS3 amazonS3;
```

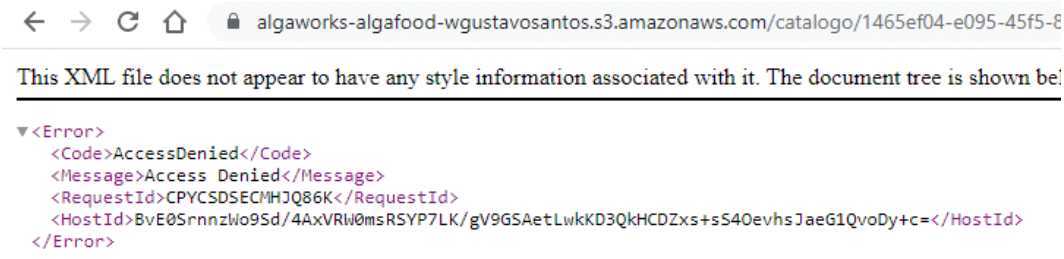
A classe S3FotoStorage precisa de um Bean da interface AmazonS3 configurado com as credenciais vistas na aula 14.22

```java
@Override
public void armazenar(NovaFoto novaFoto) {
    final String bucket = storageProperties.getS3().getBucket();
    final String caminhoArquivo = getCaminhoArquivo(novaFoto.getNomeArquivo());
    final InputStream inputStream = novaFoto.getInputStream();
    final ObjectMetadata objectMetadata = new ObjectMetadata();

    final PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, caminhoArquivo, inputStream, objectMetadata);

    try {
        amazonS3.putObject(putObjectRequest);
    } catch (Exception e){
        throw new StorageException("Não foi possível enviar arquivo para Amazon S3.", e);
    }
}
```

o bean possui um método chamado putObject que recebe um objeto PutObjectRequest, que pode ser instanciado passando o bucket, o caminho do arquivo dentro do bucket, o inputStream do arquivo a ser salvo, e um objectMetadata.

```java
1 usage
private String getCaminhoArquivo(String nomeArquivo) {
    /*catalogo/nome_da_foto.jpg*/
    return String.format("%s/%s", storageProperties.getS3().getDiretorioFotos(), nomeArquivo);
}
```

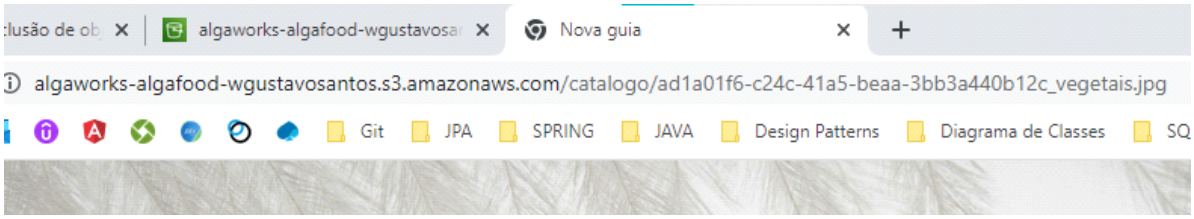
Ao tentar visualizar a foto pela url do arquivo, ocorre um erro de acesso negado



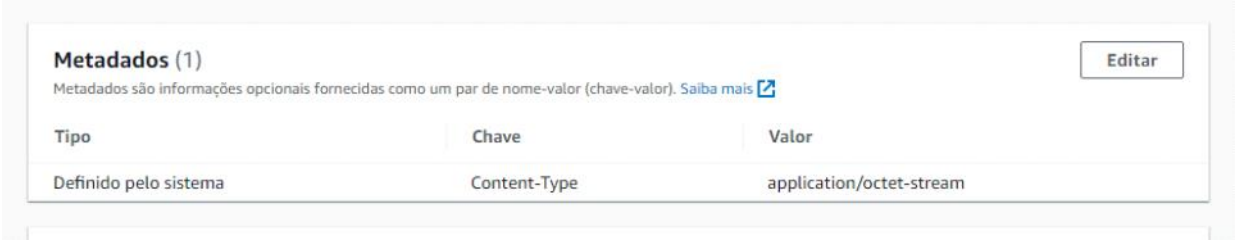
e precisamos adicionar permissões para leitura usando withCannedAcl passando um objeto CannedAccessControlList.publicRead;

```java
1 usage
@Override
public void armazenar(NovaFoto novaFoto) {
    final String bucket = storageProperties.getS3().getBucket();
    final String caminhoArquivo = getCaminhoArquivo(novaFoto.getNomeArquivo());
    final InputStream inputStream = novaFoto.getInputStream();
    final ObjectMetadata objectMetadata = new ObjectMetadata();
    final CannedAccessControlList publicRead = CannedAccessControlList.PublicRe

    final PutObjectRequest putObjectRequest = new PutObjectRequest
            (bucket, caminhoArquivo, inputStream, objectMetadata);

    putObjectRequest.withCannedAcl(publicRead);

    try {
        amazonS3.putObject(putObjectRequest);
    } catch (Exception e){
```

entretanto, o navegador faz o download da imagem



pois o objeto enviado foi configurado por padrão com o contentType:application/octet-stream. objeto enviado >> propriedades >> metadados



e precisamos adicionar um content type no momento do envio do arquivo e deixarmos dinâmicos.

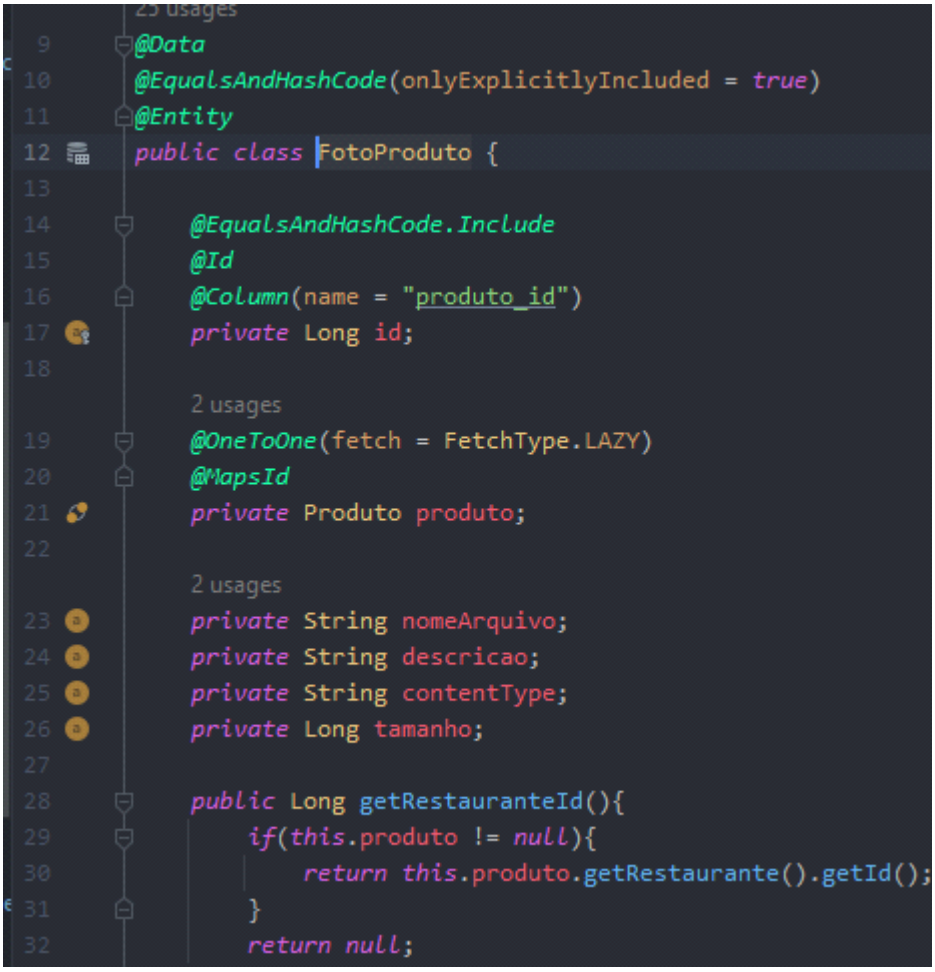
```
@Override
public void armazenar(NovaFoto novaFoto) {
    final String bucket = storageProperties.getS3().getBucket();
    final String caminhoArquivo = getCaminhoArquivo(novaFoto.getNomeArquivo());
    final InputStream inputStream = novaFoto.getInputStream();
    final String contentType = novaFoto.getContentType();
    final ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setContentType(contentType);
    final CannedAccessControlList publicRead = CannedAccessControlList.PublicRead;

    final PutObjectRequest putObjectRequest = new PutObjectRequest
            (bucket, caminhoArquivo, inputStream, objectMetadata);
```

foi adicionado um novo atributo em NovaFoto que guarda o contentType do arquivo para setar em um objeto ObjectMetadata .

```java
package com.algaworks.algafood.domain.repository;

import lombok.Builder;
import lombok.Getter;

import java.io.InputStream;

10 usages  2 implementations
public interface FotoStorageService {

    1 usage  2 implementations
    void armazenar(NovaFoto novaFoto);

    2 usages  2 implementations
    void remover(String nomeArquivo);

    1 usage  2 implementations
    InputStream recuperar(String nomeArquivo);

    5 usages
    @Getter
    @Builder
    class NovaFoto {
        private String nomeArquivo;
        private String contentType;
        private InputStream inputStream;
    }
}
```

```java
final FotoProduto fotoSalva = produtoRepository.save(fotoProduto);


FotoStorageService.NovaFoto foto = FotoStorageService.NovaFoto.builder()
        .nomeArquivo(fotoSalva.getNomeArquivo())
        .contentType(fotoSalva.getContentType())
        .inputStream(inputStream).build();

fotoStorageService.armazenar(foto);
```

na camada de serviço em CatalogoFotoProdutoService instanciamos um objeto NovaFoto a partir da foto persistida no banco de dados


25 usages
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@Entity
public class FotoProduto {
@EqualsAndHashCode.Include
@Id
@Column(name = "produto_id")
private Long id;
2 usages
@OneToOne(fetch = FetchType.LAZY)
@MapsId
private Produto produto;
2 usages
private String nomeArquivo;
private String descricao;
private String contentType;
private Long tamanho;
public Long getRestauranteId(){
if(this.produto != null){
return this.produto.getRestaurante().getId();
}
return null;

14.24. Desafio: Implementando a exclusão de objetos do bucket da Amazon S3

quarta-feira, 5 de abril de 2023 07:59

```
2 usages
@Override
public void remover(String nomeArquivo) {
    String caminhoArquivo = getCaminhoArquivo(nomeArquivo);
    String bucket = storageProperties.getS3().getBucket();
    DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest(bucket, caminhoArquivo);
    amazonS3.deleteObject(deleteObjectRequest);
}
```

14.25. Implementando a recuperação de foto no serviço de storage do S3

quarta-feira, 5 de abril de 2023 08:09

- Alterar o retorno do método na interface FotoStorageService para o tipo FotoRecuperada

```java
package com.algaworks.algafood.domain.repository;

import lombok.Builder;
import lombok.Getter;

import java.io.InputStream;

11 usages  2 implementations
public interface FotoStorageService {

    1 usage  2 implementations
    void armazenar(NovaFoto novaFoto);

    2 usages  2 implementations
    void remover(String nomeArquivo);

    1 usage  2 implementations
    FotoRecuperada recuperar(String nomeArquivo);
```

```java
FotoRecuperada recuperar(String nomeArquivo);

5 usages
@Getter
@Builder
class NovaFoto {
    private String nomeArquivo;
    private String contentType;
    private InputStream inputStream;
}

6 usages
@Builder
@Getter
class  FotoRecuperada {
    1 usage
    private InputStream inputStreamFoto;
    1 usage
    private String urlFoto;

    1 usage
    public boolean temUrl(){
        return urlFoto != null;
    }

    public boolean temInputStream(){
        return inputStreamFoto != null;
    }
}
```

- Refatorar os trechos de códigos que usavam o retorno tipo InputStream para FotoRecuperada
  - A ideia é encapsular o tipo de objeto recuperado, se o objeto estar localmente armazenado, o InputStream do objeto é obtido, se o objeto estar na nuvem, se obtém a URL do objeto.
- Em S3FotoStorageService

```java
1 usage
@Override
public FotoRecuperada recuperar(String nomeArquivo) {
    final String bucket = storageProperties.getS3().getBucket();
    final String caminhoArquivo = getCaminhoArquivo(nomeArquivo);
    final URL url = amazonS3.getUrl(bucket, caminhoArquivo);

    return FotoRecuperada.builder().urlFoto(url.toString()).build();
}
```

caminhoArquivo: catalogo/foto.jpg

```java
@GetMapping
public ResponseEntity<?> servirFoto(@PathVariable Long restauranteId,
                                    @PathVariable Long produtoId,
                                    @RequestHeader(name = "accept") String accepHeader

        FotoProduto fotoProduto = catalogoFotoProdutoService.buscarOuFalhar(restauranteId, produtoId);
        MediaType mediaTypeFotoProduto = MediaType.parseMediaType(fotoProduto.getContentType());
        List<MediaType> mediaTypesAceitas = MediaType.parseMediaTypes(accepHeader);

        verificarCompatibilidadeMediaType(mediaTypeFotoProduto, mediaTypesAceitas);

        try{
            final FotoStorageService.FotoRecuperada fotoRecuperada = fotoStorageService.recuperar(fotoPr

            if(fotoRecuperada.temUrl()){
                return ResponseEntity
                        .status(HttpStatus.FOUND)
                        .header(HttpHeaders.LOCATION, fotoRecuperada.getUrlFoto()).build();

            } else{
                return ResponseEntity
                        .ok()
                        .contentType(mediaTypeFotoProduto)
                        .body(new InputStreamResource(fotoRecuperada.getInputStreamFoto()));
            }
    } catch(EntidadeNaoEncontradaException e){
        return ResponseEntity.notFound().build();
    }
}
```

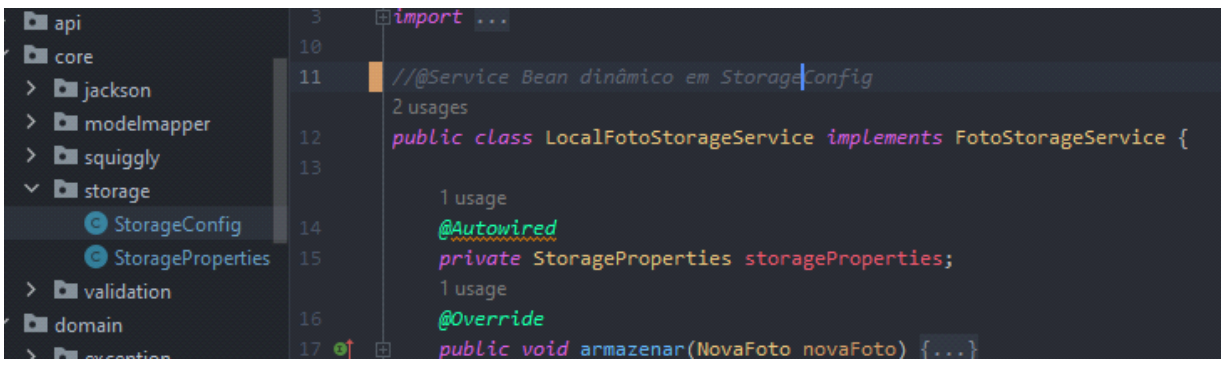
]

14.26. Selecionando a implementação do serviço de storage de fotos

quarta-feira, 5 de abril de 2023 09:26

- Deixando mais dinâmico o Bean de instanciamento do tipo de armazenamento para Local ou Nuvem

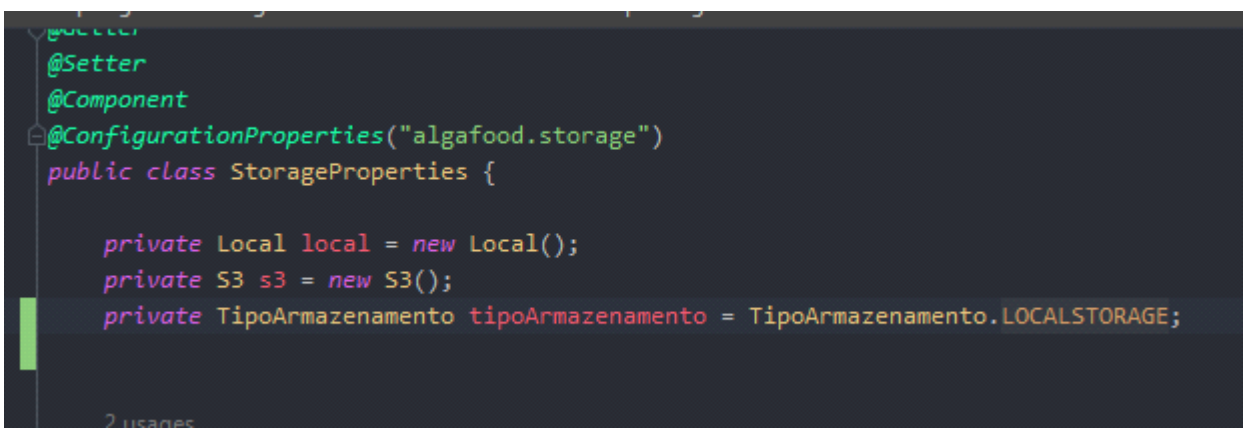
- Retirar as anotações de @Service das implementações da interface FotoStorageService

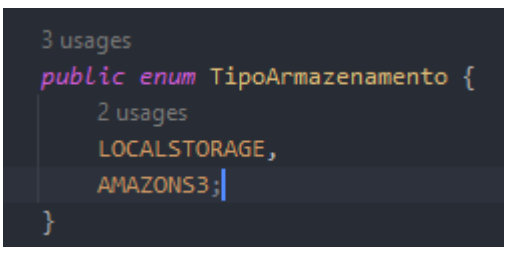


- Em StorageConfig

retornar a interface aplicando o Polimorfismo, como ambas as classes são um FotoStorageService, ambos entram no conceito "São um FotoStorageService"

Para fazer a condição de qual tipo de Bean de armazenamento queremos na injeção de dependência, foi criado um tipo enumerado como atributo em StorageProperties





E podemos facilmente atribuir o tipo no arquivo de configuração