

Refatoração e testes de software

André Takeshi Takara - RA: 12522170086

Marcelo Henrique da Silva Ventura - RA: 12522128126

Andy Hyong Tea Choi Youn - RA: 12522142446

Felipe Daura Lanzzone Ferreira - RA: 1252214120

Principais refatorações:

- Criação do controle de exceções;
- Remoção de classe com mesmas variáveis e métodos;
- Refatoração de classes record;
- Design patterns: Chain of responsibility para aplicar todas as validações;
- Criamos uma DTO para o retorno de todos os controllers;
- Polimorfismo (GRASP);
- Controller (GRASP);
- Creator (GRASP);
- Information Expert (GRASP);
- Indirection (GRASP);
- High Cohesion (GRASP);
- Low Coupling (GRASP);
- SRP – Princípio da responsabilidade única (SOLID);
- OCP – Princípio do aberto/fechado (SOLID);
- DIP – Princípio da inversão de dependência (SOLID);
- LSP - Princípio da Substituição de Liskov

```

1 package med.voll.api.infra.exceptions;
2
3 > import ...
15
2 usages
16 @RestControllerAdvice
17 public class TratadorDeErros {
18
19     1 usage
20     @ExceptionHandler(EntityNotFoundException.class)
21     public ResponseEntity<String> tratarErro404() { return ResponseEntity.notFound().build(); }
23
24     no usages
25     @ExceptionHandler(MethodArgumentNotValidException.class)
26     public ResponseEntity<List<DadosErroValidacao>> tratarErro400(MethodArgumentNotValidException ex) {
27         var erros = ex.getFieldErrors();
28         return ResponseEntity.badRequest().body(erros.stream().map(DadosErroValidacao::new).toList());
29     }
30
31     1 usage
32     @ExceptionHandler(HttpMessageNotReadableException.class)
33     public ResponseEntity<String> tratarErro400(HttpMessageNotReadableException ex) {
34         return ResponseEntity.badRequest().body(ex.getMessage());
35     }
36
37     1 usage
38     @ExceptionHandler(Exception.class)
39     public ResponseEntity<String> tratarErro500(Exception ex) {
40         return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Erro: " + ex.getLocalizedMessage());
41     }
42
43     1 usage
44     @ExceptionHandler(ValidacaoException.class)
45     public ResponseEntity<String> tratarErroRegraDeNegocio(Exception ex) {
46         return ResponseEntity.badRequest().body(ex.getMessage());
47     }
48
49     @Getter
50     @Setter
51     public class DadosErroValidacao {
52
53         String campo;
54         String mensagem;
55
56         1 usage
57         public DadosErroValidacao(FieldError erro) {
58             this(erro.getField(), erro.getDefaultMessage());
59         }
60
61         1 usage
62         public DadosErroValidacao(String field, String defaultMessage) {
63             this(field, defaultMessage);
64         }
65     }
66 }

```

infra.exceptions 100% classes, 92% lines covered
 ➔ TratadorDeErros 90% methods, 91% lines covered
 ➔ ValidacaoException 100% methods, 100% lines covered

```

ValidacaoException.java x
1 package med.voll.api.infra.exceptions;
2
3 public class ValidacaoException extends RuntimeException {
4     public ValidacaoException(String message) { super(message); }
5 }
6
7
8
9
10

```

Controle de exceções nas classes

```

58 if (medico == null) {
59     throw new ValidacaoException("Não existe médico disponível nesta data");
60 }
61
62
63 // Validação
64 @
65 public DadosDetalhamentoConsulta agendar(DadosAgendamentoConsulta dados) {
66     if (!pacienteRepository.existsById(dados.getIdPaciente())) {
67         throw new ValidacaoException("Id do paciente não existe");
68     }
69     if (dados.getIdMedico() != null && !medicoRepository.existsById(dados.getIdMedico())) {
70         throw new ValidacaoException("Id do medico não existe");
71     }
72
73     if (dados.getEspecialidade() == null) {
74         throw new ValidacaoException("Especialidade é obrigatório quando médico não for escolhido");
75     }
76
77     if (!consultaRepository.existsById(dados.getIdConsulta())) {
78         throw new ValidacaoException("Id da consulta informado não existe!");
79     }
80 }
81

```

```

1 package med.voll.api.domain.DTO.endereco;
2
3 > import ...
4
5 16 usages
6 public record DadosEndereco(
7     3 usages
8     @NotBlank
9     String logradouro,
10    3 usages
11    @NotBlank
12    @Pattern(regexp = "\\d{8}")
13    String cep,
14    3 usages
15    @NotBlank
16    String cidade,
17    3 usages
18    @NotBlank
19    String uf,
20    3 usages
21    String complemento,
22    3 usages
23    String numero) {
24 }
25

```



```

1 package med.voll.api.domain.entity;
2
3 > import ...
4
5 10 usages
6 @Embeddable
7 @Getter
8 @Setter
9 @NoArgsConstructor
10 @AllArgsConstructor
11 public class Endereco {
12
13     String logradouro;
14     String bairro;
15     String cep;
16     String cidade;
17     String uf;
18     String complemento;
19     String numero;
20
21     public Endereco(Endereco endereco) {
22         logradouro = endereco.getLogradouro();
23         bairro = endereco.getBairro();
24         cep = endereco.getCep();
25         cidade = endereco.getCidade();
26         uf = endereco.getUf();
27         complemento = endereco.getComplemento();
28         numero = endereco.getNumero();
29     }
30
31     5 usages
32     public void atualizarInformacoes(Endereco endereco) {
33         if(endereco.getLogradouro() != null) {
34             logradouro = endereco.getLogradouro();
35         }
36         if(endereco.getBairro() != null) {
37             bairro = endereco.getBairro();
38         }
39         if(endereco.getCep() != null) {
40             cep = endereco.getCep();
41         }
42         if(endereco.getCidade() != null) {
43             cidade = endereco.getCidade();
44         }
45         if(endereco.getUf() != null) {
46             uf = endereco.getUf();
47         }
48         if(endereco.getComplemento() != null) {
49             complemento = endereco.getComplemento();
50         }
51         if(endereco.getNumero() != null) {
52             numero = endereco.getNumero();
53         }
54     }
55
56     @Override
57     public boolean equals(Object o) {
58         if (this == o) return true;
59         if (o == null || getClass() != o.getClass()) return false;
60         Endereco endereco = (Endereco) o;
61         return Objects.equals(logradouro, endereco.logradouro) && Objects.equals(bairro, endereco.bairro) && Objects.equals(cep, endereco.cep) && Objects.equals(cidade,
62             endereco.cidade) && Objects.equals(uf, endereco.uf) && Objects.equals(complemento, endereco.complemento) && Objects.equals(numero, endereco.numero);
63     }
64
65     @Override
66     public int hashCode() {
67         return Objects.hash(logradouro, bairro, cep, cidade, uf, complemento, numero);
68     }
69 }
70

```

```

1 package med.voll.api.service;
2
3 > import ...
4
21
22 9 usages
23 @Setter
24 @Getter
25 @Service
26 public class AgendaDeConsultasService {
27
28     @Autowired
29     private ConsultaRepository consultaRepository;
30
31     @Autowired
32     private MedicoRepository medicoRepository;
33
34     @Autowired
35     private PacienteRepository pacienteRepository;
36
37     @Autowired
38     private List<ValidadorAgendamentoConsulta> validadores;
39
40     @Autowired
41     private List<ValidadorCancelamentoDeConsulta> validadoresCancelamento;
42
43     // Validação
44     @ public DadosDetalhamentoConsulta agendar(DadosAgendamentoConsulta dados) {
45         if (!pacienteRepository.existsById(dados.getIdPaciente())) {
46             throw new ValidacaoException("Id do paciente não existe");
47         }
48         if (dados.getIdMedico() != null && !medicoRepository.existsById(dados.getIdMedico())) {
49             throw new ValidacaoException("Id do medico não existe");
50         }

```

```

36 @Autowired
37 private List<ValidadorAgendamentoConsulta> validadores;
38
39 @Autowired
40 private List<ValidadorCancelamentoDeConsulta> validadoresCancelamento;
41

```

```

1 package med.voll.api.validation.consulta.cancelar;
2
3 import med.voll.api.domain.DTO.consulta.DadosCancelamentoConsulta;
4
5 14 usages 1 implementation
6 public interface ValidadorCancelamentoDeConsulta {
7
8     1 implementation
9     void validar(DadosCancelamentoConsulta dados);
10 }

```

```

no usages
29 @PostMapping
30 @Transactional
31 @
public ResponseEntity<Medico> cadastrar(@RequestBody @Valid DadosCadastroMedico dadosCadastroMedico, UriComponentsBuilder uriBuilder){
32     Medico medico = new Medico(dadosCadastroMedico);
33     repository.save(medico);
34     URI uri = uriBuilder.path("/medicos/{id}").buildAndExpand(medico.getId()).toUri();
35
36     return ResponseEntity.created(uri).body(medico);
37 }
38

```

```

1 usage
28 @PostMapping
29 @Transactional
30 @
public ResponseEntity<DadosDetalhamentoMedico> cadastrar(@RequestBody @Valid DadosCadastroMedico dadosCadastroMedico, UriComponentsBuilder uriBuilder){
31     Medico medico = new Medico(dadosCadastroMedico);
32     repository.save(medico);
33     URI uri = uriBuilder.path("/medicos/{id}").buildAndExpand(medico.getId()).toUri();
34
35     return ResponseEntity.created(uri).body(new DadosDetalhamentoMedico(medico));
36 }

```

```

1 package med.voll.api.controller;
2
3 > import ...
4
5 3 usages
6 @RestController
7 @RequestMapping("/consultas")
8 public class ConsultaController {
9
10     2 usages
11     @Autowired
12     private AgendaDeConsultasService service;
13
14     @PostMapping
15     @Transactional
16     public ResponseEntity<DadosDetalhamentoConsulta> agendar(@RequestBody @Valid DadosAgendamentoConsulta dados) {
17
18         DadosDetalhamentoConsulta dto = service.agendar(dados);
19         return ResponseEntity.ok(dto);
20     }
21
22     @DeleteMapping
23     @Transactional
24     public ResponseEntity cancelar(@RequestBody @Valid DadosCancelamentoConsulta dados) {
25         service.cancelar(dados);
26         return ResponseEntity.noContent().build();
27     }
28 }
29
30 }
31
32
33
34
35

```

- Princípio de Information Expert;
- Princípio de Controller:
- High Cohesion;
- Low Coupling;

```

25 public class AgendaDeConsultasService {
26
27     @Autowired
28     private ConsultaRepository consultaRepository;
29
30     @Autowired
31     private MedicoRepository medicoRepository;
32
33     @Autowired
34     private PacienteRepository pacienteRepository;
35
36     @Autowired
37     private List<ValidadorAgendamentoConsulta> validadores;
38
39     @Autowired
40     private List<ValidadorCancelamentoDeConsulta> validadoresCancelamento;
41
42
43     // Validação
44     @Transactional
45     public DadosDetalhamentoConsulta agendar(DadosAgendamentoConsulta dados) {
46         if (!pacienteRepository.existsById(dados.getIdPaciente())) {
47             throw new ValidacaoException("Id do paciente não existe");
48         }
49         if (dados.getIdMedico() != null && !medicoRepository.existsById(dados.getIdMedico())) {
50             throw new ValidacaoException("Id do medico não existe");
51         }
52
53         validadores.forEach(v -> v.validar(dados));
54
55         Paciente paciente = pacienteRepository.getReferenceById(dados.getIdPaciente());
56         Medico medico = escolherMedico(dados);
57         if (medico == null) {
58             throw new ValidacaoException("Não existe médico disponível nesta data");
59         }
60         Consulta consulta = new Consulta(id: null, medico, paciente, dados.getData(), cancelada: false);
61         consultaRepository.save(consulta);
62         return new DadosDetalhamentoConsulta(consulta);
63     }
64
65 }

```

- Princípio de Information Expert;

- High Cohesion;

- Indirection;

- Creator

```

Consulta consulta = new Consulta(id: null, medico, paciente, dados.getData(), cancelada: false);
consultaRepository.save(consulta);
return new DadosDetalhamentoConsulta(consulta);

```



```

3  > import ...
10
11  @Entity(name = "Consulta")
12  @Table(name = "consultas")
13  @Getter
14  @Setter
15  @NoArgsConstructor
16  @AllArgsConstructor
17  @EqualsAndHashCode(of = "id")
18  public class Consulta {
19
20      @Id
21      @GeneratedValue(strategy = GenerationType.IDENTITY)
22      private Long id;
23
24      @ManyToOne(fetch = FetchType.EAGER)
25      @JoinColumn(name = "medico_id")
26      private Medico medico;
27
28      @ManyToOne(fetch = FetchType.EAGER)
29      @JoinColumn(name = "paciente_id")
30      private Paciente paciente;
31
32      private LocalDateTime data;
33
34      @Column(name = "motivo_cancelamento")
35      @Enumerated(EnumType.STRING)
36      private MotivoCancelamento motivoCancelamento;
37      private boolean cancelada;
38
39      public Consulta(Long id, Medico medico, Paciente paciente, LocalDateTime data, boolean cancelada) {
40      }
41
42      public void cancelar(MotivoCancelamento motivo) {
43          this.motivoCancelamento = motivo;
44      }
45      > public boolean isCancelada() { return cancelada; }
46
47      4 usages
48
49      2 usages
50      > public void setCancelada(boolean cancelada) { this.cancelada = cancelada; }
51
52  }

```

- Princípio de Information Expert;

- High Cohesion;

```

1 package med.voll.api.validation.consulta.agendar;
2
3 > import ...
4
5
6
7
8
9
10 1 usage
11 @Component
12 public class ValidadorMedicoAtivo implements ValidadorAgendamentoConsulta{
13     1 usage
14     @Autowired
15     private MedicoRepository repository;
16     public void validar(DadosAgendamentoConsulta dados){
17         if(dados.getIdMedico() == null){
18             return;
19         }
20         Boolean medicoEstaAtivo = repository.findAtivoById(dados.getIdMedico());
21         if(!medicoEstaAtivo){
22             throw new ValidacaoException("Consulta não pode ser agendada com médico excluído");
23         }
24     }
25 }

```

- Princípio de Polymorphism;
- Low Coupling;
- High Cohesion;

- Protected Variations

```

package med.voll.api.validation.consulta.agendar;

import med.voll.api.domain.DTO.consulta.DadosAgendamentoConsulta;

1 usage 6 implementations
public interface ValidadorAgendamentoConsulta {
    6 implementations
    void validar(DadosAgendamentoConsulta dados);
}

```

```

@RestController
@RequestMapping("/pacientes")
public class PacienteController {

    5 usages
    @Autowired
    private PacienteRepository repository;

    1 usage
    @PostMapping
    @Transactional
    public ResponseEntity<DadosDetalhamentoPaciente> cadastrar(@RequestBody @Valid DadosCadastroPaciente dados, UriComponentsBuilder uriBuilder) {
        Paciente paciente = new Paciente(dados);
        repository.save(paciente);
        URI uri = uriBuilder.path("/pacientes/{id}").buildAndExpand(paciente.getId()).toUri();
        // Acrescentar DadosDetalhamento
        return ResponseEntity.created(uri).body(new DadosDetalhamentoPaciente(paciente));
    }

    1 usage
    @GetMapping
    public ResponseEntity<Page<DadosListagemPaciente>> listar(@PageableDefault(page = 0, size = 10, sort = { "nome" }) Pageable paginacao) {
        Page<DadosListagemPaciente> page = repository.findAllByAtivoTrue(paginacao).map(DadosListagemPaciente::new);
        return ResponseEntity.ok(page);
    }

    1 usage
    @PutMapping
    @Transactional
    public ResponseEntity<DadosDetalhamentoPaciente> atualizar(@RequestBody @Valid DadosAtualizacaoPaciente dadosPaciente) {
        Paciente paciente = repository.getReferenceById(dadosPaciente.id());
        paciente.atualizarInformacoes(dadosPaciente);
        // Acrescentar DadosDetalhamento
        return ResponseEntity.ok(new DadosDetalhamentoPaciente(paciente));
    }
}

```

- SRP – Princípio da responsabilidade única (SOLID);

```

@DeleteMapping("/{id}")
@Transactional
public ResponseEntity<Void> remover(@PathVariable Long id) {
    // Paciente paciente
    var paciente = repository.getReferenceById(id);
    paciente.inativar();
    return ResponseEntity.noContent().build();
}

1 usage
@GetMapping("/{id}")
public ResponseEntity<DadosDetalhamentoPaciente> detalhar(@PathVariable Long id) {
    var paciente = repository.getReferenceById(id);
    // Acrescentar DadosDetalhamento
    return ResponseEntity.ok(new DadosDetalhamentoPaciente(paciente));
}

```

@Component

```
public class CreditCardPayment implements Pagamento{

    1 usage
    private final ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
    no usages
    private final Validator validator = factory.getValidator();

    1 usage
    @Autowired
    private PacienteRepository pacienteRepository;

    1 usage
    @Autowired
    private MedicoRepository medicoRepository;

    1 usage
    @Autowired
    private List<ValidadorCancelamentoDeConsulta> validadoresCancelamento;

    2 usages
    @Autowired
    private ConsultaRepository consultaRepository;

    1 usage
    CreditCardPayment() {
    }

    1 override
    public void validar(DadosCancelamentoConsulta dados) {

        if (!consultaRepository.existsById(dados.getIdConsulta())) {
            throw new ValidacaoException("Id da consulta informado não existe!");
        }

        var consulta = consultaRepository.getReferenceById(dados.getIdConsulta());
        if (consulta == null) {
            throw new ValidacaoException("Consulta não encontrada para o ID fornecido");
        }
    }
}
```

- OCP – Princípio do aberto/fechado (SOLID);

```
validadoresCancelamento.forEach(v -> v.validar(dados));
consulta.cancelar(dados.getMotivo());
}

2 usages
public void pagar(DadosAgendamentoConsulta dados, double valorConsulta) {
    Paciente paciente = pacienteRepository.findById(dados.getIdPaciente()).get();
    Medico medico = medicoRepository.findById(dados.getIdMedico()).get();
    Pagar pagamento = new Pagar(paciente.getNome(), medico.getNome(), dados.getData(), valorConsulta);
}
```

```

package med.voll.api.validation.consulta.agendar;

import med.voll.api.domain.DTO.consulta.DadosAgendamentoConsulta;

13 usages 6 implementations
public interface ValidadorAgendamentoConsulta {
    6 implementations
    void validar(DadosAgendamentoConsulta dados);
}

```

- DIP – Princípio da inversão de dependência (SOLID);

- LSP - Princípio da Substituição de Liskov

```

package med.voll.api.validation.consulta.agendar;




import ...

1 usage
@Component("ValidadorHorarioAntecedenciaAgendamento")
public class ValidadorHorarioAntecedencia implements ValidadorAgendamentoConsulta{
    public void validar(DadosAgendamentoConsulta dados){
        LocalDateTime dataConsulta = dados.getData();
        LocalDateTime now = LocalDateTime.now();
        long diferencaMinutos = Duration.between(now, dataConsulta).toMinutes();
        if (diferencaMinutos < 30){
            throw new ValidacaoException("Consulta deve ser agendada com antecedencia de 30 minutos");
        }
    }
}

```

Principais testes:

- PacienteControllerUnitarioTest;
- MedicoControllerUnitarioTest;
- ExameIntegrationTest;
- MedicoRepositoryIntegrationTest;

Coverage java in api ×			
			
Element	Class, % ^	Method, %	Line, %
▼  all	100% (43/43)	88% (194/220)	90% (400/440)
>  med	100% (43/43)	88% (194/220)	90% (400/440)

PacienteControllerUnitarioTest

```
52 @Test
53 void cadastrar() {
54     DadosCadastroPaciente dadosCadastroPaciente = new DadosCadastroPaciente(nome: "Manuel", email: "manuel@hotmail.com", telefone: "115545445",
55     cpf: "4541551554", new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
56     uf: "SP", complemento: "casa 8", numero: "98"));
57     Paciente paciente = new Paciente(dadosCadastroPaciente);
58     when(repository.save(paciente)).thenReturn(paciente);
59     UriComponentsBuilder uriBuilder = UriComponentsBuilder.newInstance();
60     ResponseEntity<DadosDetalhamentoPaciente> responseEntity = pacienteController.cadastrar(dadosCadastroPaciente, uriBuilder);
61     Paciente paciente1 = new Paciente(responseEntity.getBody());
62     assertEquals( expected: 201, responseEntity.getStatusCodeValue());
63     assertEquals(paciente, paciente1);
64 }
65
```

```
66 @Test
67 void listar() {
68     Pageable pageable = Pageable.unpaged();
69     when(repository.findAllByAtivoTrue((pageable))).thenReturn(new PageImpl<>(Collections.emptyList()));
70     ResponseEntity<Page<DadosListagemPaciente>> responseEntity = pacienteController.listar(pageable);
71     assertEquals( expected: 200, responseEntity.getStatusCodeValue());
72     assertThat(responseEntity.getBody().getContent()).isEmpty();
73 }

```

```
75 @Test
76 void atualizar() {
77     DadosCadastroPaciente dadosCadastroPaciente = new DadosCadastroPaciente(nome: "Manuel", email: "manuel@hotmail.com", telefone: "115545445",
78     cpf: "4541551554", new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
79     uf: "SP", complemento: "casa 8", numero: "98"));
80     DadosAtualizacaoPaciente dadosAtualizacaoPaciente = new DadosAtualizacaoPaciente(id: 15L, nome: "Lucas", telefone: "5585245", new Endereco());
81     Paciente paciente = new Paciente(dadosCadastroPaciente);
82     when(repository.getReferenceById(anyLong())).thenReturn(paciente);
83     when(repository.save(any(Paciente.class))).thenReturn(paciente);
84     ResponseEntity<DadosDetalhamentoPaciente> responseEntity = pacienteController.atualizar(dadosAtualizacaoPaciente);
85     Paciente paciente1 = new Paciente(responseEntity.getBody());
86     assertEquals( expected: 200, responseEntity.getStatusCodeValue());
87     assertEquals(paciente, paciente1);
88 }

```

```
90 @Test
91 void remover() {
92     Long pacienteId = 1L;
93     when(repository.getReferenceById(anyLong())).thenReturn(new Paciente());
94     ResponseEntity<Void> responseEntity = pacienteController.remover(pacienteId);
95     assertEquals( expected: 204, responseEntity.getStatusCodeValue());
96     verify(repository).getReferenceById(pacienteId);
97 }
98
99 @Test
100 void detalhar() {
101     Long pacienteId = 1L;
102     when(repository.getReferenceById(anyLong())).thenReturn(new Paciente());
103     ResponseEntity<DadosDetalhamentoPaciente> responseEntity = pacienteController.detalhar(pacienteId);
104     assertEquals( expected: 200, responseEntity.getStatusCodeValue());
105 }

```

```
private DadosCadastroPaciente dadosPaciente(){
    return new DadosCadastroPaciente(nome: "Paulo", email: "paulinho@gmail.com", telefone: "55522552", cpf: "963852711", dadosEndereco());
}
1 usage
private Endereco dadosEndereco(){
    return new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
        uf: "SP", complemento: "casa 8", numero: "98");
}
}
```

```

@Test
void cadastrar() {
    DadosCadastroMedico dadosCadastroMedico = new DadosCadastroMedico( nome: "Teste", email: "wjhhuh@gmail.com", telefone: "11525255", crm: "w48d54w45f",
        Especialidade.CARDIOLOGIA, new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
            uf: "SP", complemento: "casa 8", numero: "98"));
    Medico medico = new Medico(dadosCadastroMedico);
    when(medicoRepository.save(medico)).thenReturn(medico);
    UriComponentsBuilder uriBuilder = UriComponentsBuilder.newInstance();
    ResponseEntity<DadosDetalhamentoMedico> responseEntity = medicoController.cadastrar(dadosCadastroMedico, uriBuilder);
    assertEquals( expected: 201, responseEntity.getStatusCodeValue());
    assertEquals(medico.getId(), responseEntity.getBody().getId());
}

@Test
void listar() {
    Pageable pageable = Pageable.unpaged();
    when(medicoRepository.findAllByAtivoTrue(pageable)).thenReturn(new PageImpl<>(Collections.emptyList()));
    ResponseEntity<Page<DadosListagemMedico>> responseEntity = medicoController.listar(pageable);
    assertEquals( expected: 200, responseEntity.getStatusCodeValue());
}

```

```

@Test
void atualizar() {
    DadosAtualizarMedico dadosAtualizarMedico = new DadosAtualizarMedico( id: 15L, nome: "Josué", telefone: "1152522", new Endereco( logradouro: "ruewewewa A",
        bairro: "baewweweirrq", cep: "wwewewe", cidade: "Sao Paulo",
        uf: "SP", complemento: "casa 8", numero: "98"));
    DadosCadastroMedico dadosCadastroMedico = new DadosCadastroMedico( nome: "Teste", email: "wjhhuh@gmail.com", telefone: "11525255", crm: "w48d54w45f",
        Especialidade.CARDIOLOGIA, new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
        uf: "SP", complemento: "casa 8", numero: "98"));
    Medico medico = new Medico(dadosCadastroMedico);
    medico.atualizarInformacoes(dadosAtualizarMedico);
    when(medicoRepository.getReferenceById(anyLong())).thenReturn(medico);
    ResponseEntity<DadosDetalhamentoMedico> responseEntity = medicoController.atualizar(dadosAtualizarMedico);
    assertEquals( expected: 200, responseEntity.getStatusCodeValue());
    assertEquals(medico.getId(), responseEntity.getBody().getId());
}

@Test
void deletar() {
    Long medicoId = 1L;
    when(medicoRepository.getReferenceById(anyLong())).thenReturn(new Medico());
    ResponseEntity<Void> responseEntity = medicoController.deletar(medicoId);
    assertEquals( expected: 204, responseEntity.getStatusCodeValue());
    verify(medicoRepository).getReferenceById(medicoId);
}

```

```

private DadosCadastroMedico dadosMedico(){
    return new DadosCadastroMedico( nome: "Lucas", email: "lucas.lucas@voll.med", telefone: "098765432",
        crm: "123456", Especialidade.CARDIOLOGIA,dadosEndereco());
}

1 usage
private Endereco dadosEndereco(){
    return new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
        uf: "SP", complemento: "casa 8", numero: "98");
}
}

```

MedicoControllerUnitarioTest


```

44 @Test
45 public void testSolicitarExame() {
46     DadosAgendamentoConsulta dadosAgendamento = new DadosAgendamentoConsulta( idMedico: 1L, idPaciente: 1L,
47         LocalDateTime.of( year: 2023, month: 12, dayOfMonth: 12, hour: 12, minute: 12), Especialidade.CARDIOLOGIA);
48     String nomeExame = "Exame de Sangue";
49     String tipoExame = "Laboratorial";
50     SolicitarExame solicitarExame = exame.solicitarExame(dadosAgendamento, nomeExame, tipoExame);
51     Paciente paciente = pacienteRepository.findById(dadosAgendamento.getIdPaciente()).get();
52     Medico medico = medicoRepository.findById(dadosAgendamento.getIdMedico()).get();
53     assertNotNull(solicitarExame);
54     assertEquals(paciente.getNome(), solicitarExame.getNomePaciente());
55     assertEquals(medico.getNome(), solicitarExame.getNomeMedico());
56     assertEquals(tipoExame, solicitarExame.getTipoExame());
57     assertEquals(dadosAgendamento.getData().plusDays(5), solicitarExame.getDataConsulta());
58     assertEquals(nomeExame, solicitarExame.getNomeExame());
59 }
60 1 usage
61 @ private DadosCadastroPaciente dadosPaciente(){
62     return new DadosCadastroPaciente( nome: "Paulo", email: "paulinho@gmail.com", telefone: "55522552", cpf: "963852711", dadosEndereco());
63 }
64 2 usages
65 @ private Endereco dadosEndereco(){
66     return new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
67         uf: "SP", complemento: "casa 8", numero: "98");
68 }
69 1 usage
70 @ private DadosCadastroMedico dadosMedico(){
71     return new DadosCadastroMedico( nome: "Lucas", email: "lucas.lucas@voll.med", telefone: "098765432",
72         crm: "123456", Especialidade.CARDIOLOGIA,dadosEndereco());
73 }

```

ExameIntegrationTest

```

21 @TestInstance(TestInstance.Lifecycle.PER_CLASS)
22 @SpringBootTest
23 public class MedicoRepositoryIntegrationTest {
24
25     5 usages
26     @Autowired
27     private MedicoRepository medicoRepository;
28     1 usage
29     @Autowired
30     private PacienteRepository pacienteRepository;
31
32     @BeforeAll    void beforeAll() {
33         pacienteRepository.save(new Paciente(dadosPaciente()));
34         medicoRepository.save(new Medico(dadosMedico()));
35     }
36
37     @Test
38     public void testFindAllByAtivoTrue() {
39         Iterable<Medico> medicos = medicoRepository.findAllByAtivoTrue(Pageable.unpaged());
40
41         assertNotNull(medicos);
42     }
43
44     @Test
45     public void testGetReferenceByIdWhereAtivoTrue() {
46         Long id = 1L;
47
48         Medico medico = medicoRepository.getReferenceByIdWhereAtivoTrue(id);
49
50         assertNotNull(medico);
51     }
52
53     @Test
54     public void testEscolherMedicoAleatorioLivreNaData() {
55         Especialidade especialidade = Especialidade.CARDIOLOGIA;
56         LocalDateTime data = LocalDateTime.now();

```

```

62 @Test
63 public void testFindAtivoById() {
64     Long id = 1L;
65
66     Boolean ativo = medicoRepository.findAtivoById(id);
67
68     assertNotNull(ativo);
69 }
70
71 1 usage
72 @ private DadosCadastroPaciente dadosPaciente(){
73     return new DadosCadastroPaciente( nome: "Paulo", email: "paulinho@gmail.com", telefone: "55522552", cpf: "963852711", dadosEndereco());
74 }
75
76 2 usages
77 @ private Endereco dadosEndereco(){
78     return new Endereco( logradouro: "rua A", bairro: "bairro", cep: "09876543", cidade: "Sao Paulo",
79         uf: "SP", complemento: "casa 8", numero: "98");
80 }
81
82 1 usage
83 @ private DadosCadastroMedico dadosMedico(){
84     return new DadosCadastroMedico( nome: "Lucas", email: "lucas.lucas@voll.med", telefone: "098765432",
85         crm: "123456", Especialidade.CARDIOLOGIA,dadosEndereco());
86 }

```

MedicoRepositoryIntegrationTest

André Takeshi Takara - RA: 12522170086

Marcelo Henrique da Silva Ventura - RA: 12522128126

Andy Hyong Tea Choi Youn - RA: 12522142446

Felipe Daura Lanzzone Ferreira - RA: 1252214120