

Documentação Teste Sicredi

Classes de domínio

Associado

Classe referente aos associados, contendo informações de ID e cpf, sendo utilizada para salvar os mesmos na tabela descrita na linha 19.

```
14  @Getter
15  @AllArgsConstructor
16  @NoArgsConstructor
17  @Builder
18  @Entity
19  @Table(name = "db_associado")
20  @SequenceGenerator(name = "seq_db_associado", sequenceName = "seq_db_associado", allocationSize = 1)
21  public class Associado {
22
23      @Id
24      @GeneratedValue(generator = "seq_db_associado")
25      @Column(name = "id", nullable = false)
26      private Integer id;
27
28      private String cpf;
29  }
30
```

Pauta

Esta é uma classe referente às pautas, contendo informações como ID, título, limite de votos para controle da finalização das sessões relacionadas a ela e data de criação. Ela é salva na tabela descrita na linha 21.

```

16  @Getter
17  @AllArgsConstructor
18  @NoArgsConstructor
19  @Builder
20  @Entity
21  @Table(name = "db_pauta")
22  @SequenceGenerator(name = "seq_db_pauta", sequenceName = "seq_db_pauta", allocationSize = 1)
23  public class Pauta {
24
25      @Id
26      @GeneratedValue(generator = "seq_db_pauta")
27      @Column(name = "id", nullable = false)
28      private Integer id;
29
30      private String titulo;
31
32      private Integer limiteVotos;
33
34      private LocalDateTime dataCriacao;
35
36  }

```

Sessão

Esta é a classe que representa as sessões, contendo informações como o ID, o ID da pauta que está sendo votada e a data limite para encerramento da sessão. Esses dados são salvos na tabela descrita na linha 21.

```

16  @Getter
17  @AllArgsConstructor
18  @NoArgsConstructor
19  @Builder
20  @Entity
21  @Table(name = "db_sessao")
22  @SequenceGenerator(name = "seq_db_sessao", sequenceName = "seq_db_sessao", allocationSize = 1)
23  public class Sessao {
24
25      @Id
26      @GeneratedValue(generator = "seq_db_sessao")
27      @Column(name = "id", nullable = false)
28      private Integer id;
29
30      private Integer idPauta;
31
32      private LocalDateTime dataLimite;
33  }

```

Valor Voto

Enum para fixação e validação dos valores de votos a serem disponibilizados e/ou recebidos.

```

6      @AllArgsConstructor
7      public enum ValorVoto {
8
9          SIM( description: "Sim"),
10         NAO( description: "Não");
11
12         @Getter
13         private final String description;
14     }

```

Voto

Esta é a classe responsável pelos votos a serem computados. Ela contém informações como o ID do voto, o ID do associado que realizou o voto, o ID da

pauta que está sendo votada e o valor do voto (Sim ou Não). Esses dados são salvos na tabela descrita na linha 19.

```
14  @Getter
15  @AllArgsConstructor
16  @NoArgsConstructor
17  @Builder
18  @Entity
19  @Table(name = "db_voto")
20  @SequenceGenerator(name = "seq_db_voto", sequenceName = "seq_db_voto", allocationSize = 1)
21  public class Voto {
22
23      @Id
24      @GeneratedValue(generator = "seq_db_voto")
25      @Column(name = "id", nullable = false)
26      private Integer id;
27
28      private Integer idAssociado;
29
30      private Integer idPauta;
31
32      private String valorVoto;
33  }
```

Repository

AssociadoRepository

Interface para utilização de métodos de interação com a tabela `"db_associado"` no banco de dados. Nenhum método declarado pois foram utilizados apenas os da interface que é estendida.

```
7  @Repository
8  public interface AssociadoRepository extends CrudRepository<Associado, Integer> {
9  }
10
```

PautaRepository

Interface para utilização de métodos de interação com a tabela `"db_pauta"` no banco de dados. Nenhum método declarado pois foram utilizados apenas os da interface que é estendida.


```
7  @Repository
8  public interface PautaRepository extends CrudRepository<Pauta, Integer> {
9  }
10
```

SessaoRepository

Interface para utilização de métodos de interação com a tabela `"db_sessao"` no banco de dados.

Métodos:

- Linha 10 (`findByIdPauta`): visa buscar uma ocorrência na tabela citada anteriormente, que contenha, na coluna `idPauta`, o valor informado no parâmetro.
- Linha 12 (`existsByIdPauta`): retorna `true` caso exista uma ocorrência na tabela que contenha, na coluna `idPauta`, o parâmetro informado. Caso contrário, retorna `false`.

```
7  @Repository
8  public interface SessaoRepository extends CrudRepository<Sessao, Integer> {
9
10     Sessao findByIdPauta(Integer idPauta);
11
12      boolean existsByIdPauta(Integer idPauta);
13 }
```

VotoRepository

Interface para utilização de métodos de interação com a tabela `"db_voto"` no banco de dados.

Métodos:

- Linha 12 (`existsByIdAssociadoAndIdPauta`): retorna `true` caso exista uma ocorrência na tabela que contenha, na coluna `idAssociado` e `idPauta`, os parâmetros informados. Caso contrário, retorna `false`
- Linha 14 (`countByIdPauta`): retorna a quantidade de ocorrências na tabela que contém, na coluna `idPauta`, o parâmetro informado.
- Linha 16 (`findAllByIdPauta`): retorna uma lista com todas as ocorrências na tabela que contém, na coluna `idPauta`, o parâmetro informado.

```

9      @Repository
10     public interface VotoRepository extends CrudRepository<Voto, Integer> {
11
12         boolean existsByIdAssociadoAndIdPauta(Integer idAssociado, Integer idPauta);
13
14         Integer countByIdPauta(Integer idPauta);
15
16         List<Voto> findAllByIdPauta(Integer idPauta);
17
18     }

```

CpfValidatorRepository

Interface para integração com serviço externo, o qual realiza a validação do cpf informado.

Métodos:

- Linha 11 (`validaCpf`): método que retorna um booleano de acordo com o retorno da chamada do serviço externo, passando o CPF como variável no path da URL.

```

7      @FeignClient(value = "cpfValidator", url = "https://valida-cpf-sicredi-production.up.railway.app")
8     public interface CpfValidatorRepository {
9
10         @GetMapping(value = "/cpf/{cpf}")
11         boolean validaCpf(@PathVariable String cpf);
12     }

```

Mapper

AssociadoMapper

Métodos:

- Linha 10 (`toDomain`): método utilizado para criar a classe domain de Associado, a partir da Request utilizada no cadastro de um novo Associado.

```
7      @Component
8      public class AssociadoMapper {
9
10     @   public Associado toDomain(CadastrarAssociadoRequest request) {
11         |       return Associado.builder()
12         |           .cpf(request.getCpf())
13         |           .build();
14     }
15 }
```

AssociadoMapper

Métodos:

- Linha 10 (`toDomain`): método utilizado para criar a classe domain de Pauta, a partir da Request utilizada no cadastro de uma nova Pauta.

```
7      @Component
8      public class PautaMapper {
9
10     @   public Pauta toDomain(final CadastrarPautaRequest request) {
11         |       return Pauta.builder()
12         |           .titulo(request.getTitulo())
13         |           .limiteVotos(request.getLimiteVotos())
14         |           .dataCriacao(request.getDataCriacao())
15         |           .build();
16     }
17 }
```

SessaoMapper

Métodos:

- Linha 12 (`toDomain`): método utilizado para criar a classe domain de Sessao, a partir da Request utilizada na criação de uma nova Sessao, bem como a data limite que é verificada na classe SessaoService.

```
9      @Component
10     public class SessaoMapper {
11
12     @      public Sessao toDomain(AbrirSessaoRequest request, LocalDateTime dataLimite) {
13         return Sessao.builder()
14             .idPauta(request.getIdPauta())
15             .dataLimite(dataLimite)
16             .build();
17     }
```

VotoMapper

Métodos:

- Linha 13 (`toDomain`): método utilizado para criar a classe domain de Voto, a partir da Request utilizada na criação de um novo Voto.
- Linha 21 (`toVencedorResponse`): método para montar a Response indicando o vencedor (ou empate) da votação.
- Linha 27 (`montaFraseVencedor`): método para estruturar a frase que irá na Response, utilizando o título da pauta e o resultado da votação.


```

10     @Component
11     public class VotoMapper {
12
13     @   public Voto toDomain(final VotoRequest request) {
14         return Voto.builder()
15             .idAssociado(request.getIdAssociado())
16             .valorVoto(ValorVoto.valueOf(request.getValorVoto()).getDescription())
17             .idPauta(request.getIdPauta())
18             .build();
19     }
20
21     @   public VencedorResponse toVencedorResponse(final Pauta pauta, String resultado) {
22         return VencedorResponse.builder()
23             .fraseVencedor(montaFraseVencedor(pauta.getTitulo(), resultado))
24             .build();
25     }
26
27     @   private String montaFraseVencedor(String titulo, String resultado) {
28         return "A pauta " + titulo + " teve o resultado: " + resultado;
29     }
30 }

```

StatusMapper

Métodos:

- Linha 9 (`toResponse`): método utilizado para criar a classe de response do Status, a partir da mensagem passada como parâmetro.

```

6     @Component
7     public class StatusMapper {
8
9     @   public StatusResponse toResponse(String mensagem) {
10         return StatusResponse.builder()
11             .status(mensagem)
12             .build();
13     }
14 }

```

Service

AssociadoService

Métodos:

- Linha 23 (`cadastrarAssociado`): método para salvar o novo Associado cadastrado.
- Linha 43 (`consultaCpf`): método que utiliza o método `consultaCpf` da classe de serviço `cpfValidatorService`, que retorna um valor boolean determinando se o cpf é válido ou não, se for `true` um valor randômico será gerado e retornado, se for `false` será retornado `null`.
- Linha 48 (`randomizaRetorno`): método que gera um valor numérico randômico entre 0 e 1, posteriormente utilizando esse valor para decidir qual Status será retornado, caso seja 0 o Status será "ABLE_TO_VOTE", caso seja 1 o Status será "UNABLE_TO_VOTE".

```
13  @Slf4j
14  @Service
15  public class AssociadoService {
16
17      @Autowired
18      private AssociadoRepository repository;
19
20      @Autowired
21      private AssociadoMapper associadoMapper;
22
23      public void cadastrarAssociado(CadastrarAssociadoRequest request) {
24
25          Associado associado = associadoMapper.toDomain(request);
26
27          log.info("Salvando associado criado.");
28
29          repository.save(associado);
30      }
```

```

41     public StatusResponse consultaCpf(String cpf) {
42         if (cpfValidatorService.verificaCpfValido(cpf)) {
43             return randomizaRetorno();
44         }
45         return null;
46     }
47
48     public StatusResponse randomizaRetorno() {
49         Random random = new Random();
50         int resultado = random.nextInt( bound: 2);
51
52         if (resultado == 0) {
53             return statusMapper.toResponse(Status.ABLE_TO_VOTE.toString());
54         }
55         return statusMapper.toResponse(Status.UNABLE_TO_VOTE.toString());
56     }
57
58
59 }

```

PautaService

Métodos:

- Linha 25 (`cadastrarPauta`): método para salvar a nova Pauta criada, utilizando um método para validar a data de criação da mesma.
- Linha 38 (`validaDataCriacao`): método para verificar se a data de criação da Pauta é válida, no caso verificando se a mesma não é anterior ao `LocalDateTime.now()`.
- Linha 46 (`getPautaById`): método que verifica a existência de uma Pauta que possua o id informado como parâmetro.
- Linha 50 (`existsById`): método para verificar se existe uma Pauta com o id passado como parâmetro.

```

15  @Slf4j
16  @Service
17  public class PautaService {
18
19      @Autowired
20      private PautaRepository repository;
21
22      @Autowired
23      private PautaMapper pautaMapper;
24
25      public void cadastrarPauta(CadastrarPautaRequest request) {
26
27          log.info("Validando data de criação da pauta.");
28
29          validaDataCriacao(request);
30
31          Pauta pauta = pautaMapper.toDomain(request);
32
33          log.info("Salvando pauta criada.");
34
35          repository.save(pauta);
36      }
37
38      @private void validaDataCriacao(CadastrarPautaRequest request) {
39
40          if (request.getDataCriacao().isBefore(LocalDate.now())) {
41              log.info("Data de criação da pauta é inválida.");
42              throw new DataCriacaoException("Data de criação inválida.");
43          }
44      }
45
46      public Optional<Pauta> getPautaById(Integer idPauta) { return repository.findById(idPauta); }
49
50      public boolean existsById(Integer idPauta) {
51          return repository.existsById(idPauta);
52      }
53  }

```

SessaoService

Métodos:

- Linha 31 (`criarSessao`): método para salvar a nova Sessao criada, utilizando os seguintes métodos para validar as informações: `verificaSessaoExiste`, `verificaDataLimite` e `verificaSeAPautaExiste`.

```

18  @Slf4j
19  @Service
20  public class SessaoService {
21
22      @Autowired
23      private SessaoRepository repository;
24
25      @Autowired
26      private PautaService pautaService;
27
28      @Autowired
29      private SessaoMapper sessaoMapper;
30
31      public void criarSessao(AbrirSessaoRequest request) {
32
33          log.info("Verificando se já existe sessão para a pauta requisitada.");
34
35          verificaSessaoExiste(request);
36
37          log.info("Verificando se a data limite é válida.");
38
39          LocalDateTime dataLimite = verificaDataLimite(request.getDataLimite());
40
41          log.info("Verificando se a pauta existe.");
42
43          verificaSeAPautaExiste(request);
44
45          Sessao sessao = sessaoMapper.toDomain(request, dataLimite);
46
47          log.info("Salvando sessão criada.");
48
49          repository.save(sessao);
50      }

```

- Linha 52 (`verificaSessaoExiste`): método que verifica se já não existe uma sessão para aquela pauta.
- Linha 60 (`verificaDataLimite`): método para verificar se foi informada uma data limite na requisição. Se a data for nula, o método definirá 1 minuto como padrão. Se a data for anterior à data atual, o método `validaDataLimite` lançará uma exceção.

- Linha 69 (`validaDataLimite`): método que realiza a validação se a data não é anterior a data atual. Caso positivo, ele lança uma exceção.
- Linha 79 (`verificaSeAPautaExiste`): método para verifica se existe uma pauta com o idPauta informado na Request, caso negativo uma exceção será lançada.

```

52 @ private void verificaSessaoExiste(AbrirSessaoRequest request) {
53
54     if (repository.existsByIdPauta(request.getIdPauta())) {
55         log.info("Já existe sessão para a pauta requisitada.");
56         throw new SessaoException("A sessão dessa pauta já existe.");
57     }
58 }
59
60 private LocalDateTime verificaDataLimite(LocalDateTime dataLimite) {
61
62     log.info("Verificando se a data limite é nula (para criação de uma default) ou se ela é inválida.");
63
64     return ofNullable(dataLimite)
65         .map(this::validaDataLimite)
66         .orElse(LocalDateTime.now().plusMinutes(1));
67 }
68
69 @ private LocalDateTime validaDataLimite(LocalDateTime request) {
70
71     if (request.isBefore(LocalDateTime.now())) {
72         log.info("Data limite é inválida.");
73         throw new DataLimiteException("Data limite não pode ser antes da data atual.");
74     }
75
76     return request;
77 }
78
79 @ private void verificaSeAPautaExiste(AbrirSessaoRequest request) {
80
81     if (!pautaService.existsById(request.getIdPauta())) {
82         throw new PautaNaoExisteException("Pauta não existe.");
83     }
84 }

```

- Linha 86 (`sessaoEncerrada`): método para verificar se a Sessao referente a Pauta a qual o id é passado como parâmetro, está encerrada ou não.
- Linha 92 (`verificaExisteSessaoParaPauta`): método para verificar se existe uma Sessao para o idPauta informado no parâmetro.

```

86     public boolean sessaoEncerrada(Integer idPauta) {
87         return ofNullable(repository.findByIdPauta(idPauta)) Optional<Sessao>
88             .map(sessao -> sessao.getDataLimite().isBefore(LocalDate.now())) Optional<Boolean>
89             .orElse( other: false);
90     }
91
92     public boolean verificaExisteSessaoParaPauta(Integer idPauta) { return repository.existsByIdPauta(idPauta); }
93
94     }
95

```

VotoService

Métodos:

- Linha 40 (`receberVoto`): método para salvar o Voto criado, utilizando os seguintes métodos para validar as informações: `validaSessaoAindaAberta` , `validaAssociadoJaVotou` e `limiteDeVotosAtingido` para verificar se já foi atingido o limite de votos permitidos naquela Pauta.

```

21  @Service
22  public class VotoService {
23
24      @Autowired
25      private VotoRepository repository;
26
27      @Autowired
28      private VotoMapper votoMapper;
29
30      @Autowired
31      private AssociadoService associadoService;
32
33      @Autowired
34      private PautaService pautaService;
35
36      @Autowired
37      private SessaoService sessaoService;
38
39      @
40      public void receberVoto(VotoRequest request) {
41
42          log.info("Validando se sessão ainda está aberta.");
43
44          validaSessaoAindaAberta(request.getIdPauta());
45
46          log.info("Verificando se o associado já votou nessa sessão.");
47
48          validaAssociadoJaVotou(request.getIdAssociado(), request.getIdPauta());
49
50          if (limiteDeVotosAtingido(request.getIdPauta())) {
51              log.info("Limite de votos já foi atingido.");
52              throw new VotoInvalidoException("Limite de votos atingido.");
53          }
54
55          Voto voto = votoMapper.toDomain(request);
56
57          log.info("Salvando voto.");
58
59          repository.save(voto);
60      }

```


- Linha 62 (`validaSessaoAindaAberta`): método que valida se o tempo da sessão já expirou. Caso positivo, uma exceção é lançada.
- Linha 70 (`validaAssociadoJaVotou`): método para validar se o associado já votou na sessão referente àquela pauta. Caso positivo, uma exceção é lançada.

```

62     private void validaSessaoAindaAberta(Integer idPauta) {
63
64         if (sessaoService.sessaoEncerrada(idPauta)) {
65             log.info("Sessão teve o tempo expirado e está encerrada.");
66             throw new DataLimiteException("Sessão de votos encerrada para essa pauta.");
67         }
68     }
69
70     private void validaAssociadoJaVotou(Integer idAssociado, Integer idPauta) {
71
72         if (repository.existsByIdAssociadoAndIdPauta(idAssociado, idPauta)) {
73             log.info("Associado já votou nessa pauta.");
74             throw new VotoInvalidoException("Associado já votou nessa pauta.");
75         }
76     }

```

- Linha 78 (`contagemVotosVencedor`): método que retorna a classe informando o vencedor da respectiva sessão. Antes disso, o método `verificaExisteSessaoParaPauta` é utilizado para verificar a existência de uma sessão para a pauta em questão, bem como se o limite de votos foi atingido (`limiteDeVotosAtingido`) ou se aquela sessão teve seu tempo esgotado (`sessaoEncerrada`). Caso alguma dessas verificações seja positiva, é realizada a contagem de votos (`contagemVotos`) e a verificação de quem foi o vencedor ou se houve empate (`verificaVencedor`). Caso contrário, é lançada uma exceção.
- Linha 105 (`contagemVotos`): método para realizar a contagem de votos que possuem o respectivo valor informado no parâmetro valorVoto.

```

78 public VencedorResponse contagemVotosVencedor(Integer idPauta) {
79
80     if (!sessaoService.verificaExisteSessaoParaPauta(idPauta)) {
81         log.info("Sessão de votos para essa pauta não existe.");
82         throw new SessaoException("Sessão de votos para essa pauta não existe.");
83     }
84
85     if (limiteDeVotosAtingido(idPauta) || sessaoService.sessaoEncerrada(idPauta)) {
86         List<Voto> todosVotosDaPauta = repository.findAllByIdPauta(idPauta);
87         Pauta pauta = pautaService.getPautaById(idPauta).orElseThrow();
88
89         log.info("Contabilizando votos 'Sim'.");
90
91         int contagemSim = contagemVotos(todosVotosDaPauta, valorVoto: "Sim");
92
93         log.info("Contabilizando votos 'Não'.");
94
95         int contagemNao = contagemVotos(todosVotosDaPauta, valorVoto: "Não");
96
97         log.info("Verificando vencedor.");
98
99         return verificaVencedor(pauta, contagemSim, contagemNao);
100     }
101     log.info("Sessão ainda em aberto, não atingiu limite de data e/ou votos.");
102     throw new VotoInvalidoException("Sessão ainda em aberto, não atingiu limite de data e/ou votos.");
103 }
104
105 @ private int contagemVotos(List<Voto> votosTotais, String valorVoto) {
106     return votosTotais
107         .stream() Stream<Voto>
108         .filter(voto -> voto.getValorVoto().equals(valorVoto))
109         .toList() List<Voto>
110         .size();
111 }

```

- Linha 113 (`verificaVencedor`): método para verificar qual valor de voto obteve maior contagem, caso sejam contagens iguais, o método passa o valor “Empate” para o mapper.
- Linha 124 (`limiteDeVotosAtingido`): método que verifica se aquela pauta já atingiu o seu limite de votos totais.

```

113     private VencedorResponse verificaVencedor(Pauta pauta, int contagemSim, int contagemNao) {
114
115         if (contagemSim > contagemNao) {
116             return votoMapper.toVencedorResponse(pauta, ValorVoto.SIM.getDescription());
117         } else if (contagemNao > contagemSim) {
118             return votoMapper.toVencedorResponse(pauta, ValorVoto.NAO.getDescription());
119         }
120
121         return votoMapper.toVencedorResponse(pauta, resultado: "Empate");
122     }
123
124     private boolean limiteDeVotosAtingido(Integer idPauta) {
125
126         Optional<Pauta> pautaASerVotada = pautaService.getPautaById(idPauta);
127
128         return pautaASerVotada.map(pauta ->
129             Objects.equals(pauta.getLimiteVotos(), repository.countByIdPauta(idPauta)))
130             .orElse( other: false);
131     }
132
133 }

```

CpfValidatorService

Métodos:

- Linha 13 (`verificaCpfValido`): método que utiliza a classe `CpfValidatorRepository` para validação do cpf informado como parâmetro.

```

6
7     @Service
8     public class CpfValidatorService {
9
10         @Autowired
11         private CpfValidatorRepository cpfValidatorRepository;
12
13         public boolean verificaCpfValido(String cpf) {
14
15             return cpfValidatorRepository.validaCpf(cpf);
16         }
17     }

```

Controller

AssociadoController

Métodos:

- Linha 28 (`cadastrarAssociado`): método POST para realizar o cadastro do Associado.
- Linha 34 (`consultaCpf`): método GET para realizar a consulta do cpf informado no path da URL, retornando o status do cpf caso seja válido, caso negativo ele retorna “not found” (status code 404).

```
19  @RestController
20  @RequestMapping("associado")
21  public class AssociadoController {
22
23      @Autowired
24      AssociadoService associadoService;
25
26      @PostMapping
27      @ResponseStatus(HttpStatus.CREATED)
28      public void cadastrarAssociado(@RequestBody final CadastrarAssociadoRequest request) {
29          associadoService.cadastrarAssociado(request);
30      }
31
32      @GetMapping("/{cpf}")
33      @ResponseStatus(HttpStatus.OK)
34      public ResponseEntity<StatusResponse> consultaCpf(@PathVariable String cpf) {
35          return Optional.ofNullable(associadoService.consultaCpf(cpf)) Optional<StatusResponse>
36              .map(cpfResponse -> ResponseEntity.ok().body(cpfResponse)) Optional<ResponseEntity<StatusResponse>>
37              .orElseGet(() -> ResponseEntity.notFound().build());
38      }
39  }
```

PautaController

Métodos:

- Linha 22 (`cadastrarPauta`): método POST para realizar o cadastro da Pauta.

```

13  @RestController
14  @RequestMapping("pauta")
15  public class PautaController {
16
17      @Autowired
18      PautaService pautaService;
19
20      @PostMapping
21      @ResponseStatus(HttpStatus.CREATED)
22      public void cadastrarPauta(@RequestBody final CadastrarPautaRequest request) {
23          pautaService.cadastrarPauta(request);
24      }
25  }

```

SessaoController

Métodos:

- Linha 22 (`abrirSessao`): método POST para realizar a abertura de uma Sessao.

```

13  @RestController
14  @RequestMapping("sessao")
15  public class SessaoController {
16
17      @Autowired
18      SessaoService sessaoService;
19
20      @PostMapping
21      @ResponseStatus(HttpStatus.CREATED)
22      public void abrirSessao(@RequestBody final AbrirSessaoRequest request) {
23          sessaoService.criarSessao(request);
24      }

```

VotoController

Métodos:

- Linha 25 (`votarPauta`): método POST votar em uma Pauta.
- Linha 31 (`resultadoPauta`): método para realizar a contagem de votos de uma pauta específica, informada através do idPauta no path da URL.

```

16  @RestController
17  @RequestMapping("voto")
18  public class VotoController {
19
20      @Autowired
21      VotoService votoService;
22
23      @PostMapping
24      @ResponseStatus(HttpStatus.CREATED)
25      public void votarPauta(@RequestBody final VotoRequest request) {
26          votoService.receberVoto(request);
27      }
28
29      @GetMapping("{idPauta}")
30      @ResponseStatus(HttpStatus.OK)
31      public VencedorResponse resultadoPauta(@PathVariable Integer idPauta) {
32          return votoService.contagemVotosVencedor(idPauta);
33      }
34  }

```

Bônus 01: Integração com sistemas externos

Foi criada e disponibilizada na nuvem uma API que realiza a validação de CPF, retornando `false` caso o CPF seja inválido ou `true` caso seja válido. Essa API foi integrada na principal para que esse valor retornado seja utilizado para informar se o CPF do Associado é **ABLE_TO_VOTE** ou **UNABLE_TO_VOTE**, ou então retornando “not found” (status code 404) caso seja um CPF inválido.

CpfService

Métodos:

- Linha 12 (`consultaCpf`): método que retorna um valor booleano de acordo com a verificação do CPF passado como parâmetro no método `verificaCpfValido`.

```

7  @Service
8  public class CpfService {
9
10     private static final Pattern CPF_PATTERN = Pattern.compile("\\d{3}\\.?.?\\d{3}\\.?.?\\d{3}-?\\d{2}");
11
12     public boolean consultaCpf(String cpf) {
13
14         return verificaCpfValido(cpf);
15     }
16

```

- Linha 17 (`verificaCpfValido`): método que valida se o CPF informado como parâmetro é válido ou não, retornando respectivamente `true` ou `false`.

```

17 public static boolean verificaCpfValido(String cpf) {
18     if (cpf == null || !CPF_PATTERN.matcher(cpf).matches()) {
19         return false;
20     }
21
22     cpf = cpf.replaceAll( regex: "[.]", replacement: "");
23
24     int sum = 0;
25     int weight = 10;
26     for (int i = 0; i < 9; i++) {
27         int digit = Character.getNumericValue(cpf.charAt(i));
28         sum += digit * weight;
29         weight--;
30     }
31
32     int firstVerifier = 11 - (sum % 11);
33     if (firstVerifier > 9) {
34         firstVerifier = 0;
35     }
36
37     sum = 0;
38     weight = 11;
39     for (int i = 0; i < 9; i++) {
40         int digit = Character.getNumericValue(cpf.charAt(i));
41         sum += digit * weight;
42         weight--;
43     }
44     sum += firstVerifier * 2;
45
46     int secondVerifier = 11 - (sum % 11);
47     if (secondVerifier > 9) {
48         secondVerifier = 0;
49     }
50
51     return cpf.charAt(9) - '0' == firstVerifier && cpf.charAt(10) - '0' == secondVerifier;
52 }

```

CpfController

Métodos:

- Linha 21 (`consultaCpf`): método GET que retorna um valor booleano de acordo com a verificação do CPF passado como parâmetro no método `consultaCpf`.


```

12  @RestController
13  @RequestMapping()
14  public class CpfController {
15
16      @Autowired
17      CpfService cpfService;
18
19      @GetMapping("/{cpf}")
20      @ResponseStatus(HttpStatus.OK)
21      public boolean consultaCpf(@PathVariable String cpf) { return cpfService.consultaCpf(cpf); }
24  }

```

Bônus 02: Teste Performance

Foi executado um teste de carga com a finalidade de verificar a performance da aplicação. Os parâmetros utilizados estão indicados na [figura 1](#), a requisição na [figura 2](#), e os resultados obtidos estão ilustrados na [figura 3](#).

- Para realização do teste de carga a aplicação foi inicializada no ambiente local, assim os limites de gastos na nuvem não são atingidos e as aplicações hospedadas nela não são bloqueadas/excluídas. Por questões de limitações de hardware, não foi possível executar o teste com “centenas de milhares de votos”.

Propriedades do Usuário Virtual

Número de Usuários Virtuais (threads): 2000

Tempo de inicialização (em segundos): 20

Contador de Iteração ☐ Infinito 1

Figura 1.

Servidor Web

Protocolo [http]: http Nome do Servidor ou IP: localhost Número da Porta: 8090

Requisição HTTP

POST Caminho: /sicredi/voto Codificação do conte

☐ Redirecionar automaticamente ☒ Seguir redireções ☒ Usar Manter Ativo (KeepAlive) ☐ Usar multipart/form-data para HTTP POST ☐ Browser-compatible headers

Parameters Body Data Files Upload

```

1  {
2    "idAssociado": ${numeroCpf},
3    "valorVoto": "NAO",
4    "idPauta": 1
5  }

```

Figura 2.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Requisição HTTP	2000	9	5	97	6,27	0,00%	99,1/sec	11,71	24,82	121,0
TOTAL	2000	9	5	97	6,27	0,00%	99,1/sec	11,71	24,82	121,0

Figura 3.

Bônus 03: Versionamento da API

Utilizaria a metodologia de versionamento que inclui a versão na URL. Podendo ter endpoints como **/api/v1/users** e **/api/v2/users**. Isso permite que as diferentes versões coexistam na mesma aplicação, oferecendo uma maneira fácil de distinguir entre elas e possibilitando que, caso seja necessário uma migração entre versões, ela ocorra de uma maneira mais gradual e sem prejudicar sistemas/serviços que consomem esses endpoints.

Também consideraria o versionamento via inclusão da versão da API no cabeçalho da solicitação. Isso permitiria manter a mesma URL para todas as versões, possivelmente exigindo menos retrabalho caso seja necessário uma migração entre elas.