```java
package br.com.padon.application.controllers;

import br.com.padon.application.models.Pertence;
import br.com.padon.application.models.Venda;
import br.com.padon.application.repositorys.
PertenceRepository;
import br.com.padon.application.repositorys.
VendaRepository;
import com.fasterxml.jackson.databind.JsonNode;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.web.bind.annotation.*;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@RestController
@CrossOrigin
@RequestMapping("/venda")
public class VendaController {

    @Autowired
    private VendaRepository venda;
    @Autowired
    private PertenceRepository pertence;
    private final SimpleDateFormat parser = new
SimpleDateFormat("dd/MM/yyyy");


    @PostMapping("/create")
    public Venda createVenda(@RequestBody JsonNode node)
throws ParseException {
        return venda.save(new Venda(
                parser.parse(node.get("data").asText()),
                node.get("status").asBoolean(),
                node.get("valor").asDouble(),
                node.get("comanda").asInt()
        ));
    }

```

```
40      @GetMapping("/get")
41      public List<Venda> getAllVendas() {
42          return venda.findAll();
43      }
44
45      @GetMapping("/gettrue")
46      public List<Venda> getAllVendasTrue() {
47          return venda.findAll().stream().filter(Venda::
    getStatusVenda).collect(Collectors.toList());
48      }
49
50      @GetMapping("/getfalse")
51      public List<Venda> getAllVendasFalse() {
52          return venda.findAll().stream().filter(v -> !v.
    getStatusVenda()).collect(Collectors.toList());
53      }
54
55      @PostMapping("/save")
56      public Venda saveVenda(@RequestBody JsonNode node)
    throws ParseException {
57          Venda vendaById = venda.findById(node.get("id").
    asInt()).orElseThrow();
58          if (!vendaById.getStatusVenda()) {
59              throw new IllegalAccessError("venda encerrada
    , alteração negada");
60          }
61          vendaById.setDataVenda(parser.parse(node.get("data
    ").asText()));
62          vendaById.setStatusVenda(node.get("status").
    asBoolean());
63          vendaById.setValorTotal(node.get("valor").asDouble
    ());
64          vendaById.setComanda(node.get("comanda").asInt());
65          return venda.save(vendaById);
66      }
67
68      @PostMapping("/byid")
69      public Optional<Venda> getVenda(@RequestBody JsonNode
    node) {
70          return venda.findById(node.get("id").asInt());
71      }
72
73      @PostMapping("/delete")
74      public boolean deleteVenda(@RequestBody JsonNode node
```

```
74 ) {
75         Venda vendaById = venda.findById(node.get("id").
   asInt()).orElseThrow();
76         if (!vendaById.getStatusVenda()) {
77             throw new IllegalAccessError("venda encerrada
   , alteração negada");
78         }
79         venda.delete(vendaById);
80         return true;
81     }
82
83     @PostMapping("/add")
84     public Pertence addProduto(@RequestBody JsonNode node
   ) {
85         int vendaId = node.get("vendaId").asInt();
86         Venda vendaById = venda.findById(vendaId).
   orElseThrow();
87         if (!vendaById.getStatusVenda()) {
88             throw new IllegalAccessError("venda encerrada
   , alteração negada");
89         }
90
91         return pertence.save(new Pertence(
92                 node.get("produtoId").asInt(),
93                 vendaId,
94                 node.get("precoTotal").asDouble(),
95                 node.get("quantidade").asInt(),
96                 node.get("precoAtual").asDouble()
97         ));
98     }
99
100    @PostMapping("/remove")
101    public boolean removeProduto(@RequestBody JsonNode
   node) {
102        int vendaId = node.get("vendaId").asInt();
103        Venda vendaById = venda.findById(vendaId).
   orElseThrow();
104        if (!vendaById.getStatusVenda()) {
105            throw new IllegalAccessError("venda encerrada
   , alteração negada");
106        }
107
108        Pertence result = pertence.getPertence(node.get("
   produtoId").asInt(), vendaId);
```

```
109         if (result == null) {
110             return false;
111         }
112 
113         pertence.delete(result);
114         return true;
115     }
116 
117     @PostMapping("/editproduto")
118     public Pertence editProduto(@RequestBody JsonNode
    node) {
119         int vendaId = node.get("vendaId").asInt();
120         Venda vendaById = venda.findById(vendaId).
    orElseThrow();
121         if (!vendaById.getStatusVenda()) {
122             throw new IllegalAccessError("venda encerrada
    , alteração negada");
123         }
124 
125         Pertence pertenceById = pertence.getPertence(node
    .get("produtoId").asInt(), vendaId);
126         pertenceById.setPrecoAtual(node.get("precoAtual"
    ).asDouble());
127         pertenceById.setPrecoTotal(node.get("precoTotal"
    ).asDouble());
128         pertenceById.setQuantidade(node.get("quantidade"
    ).asInt());
129         return pertence.save(pertenceById);
130     }
131 }
132 
```