1. Visão Geral

- O sistema de locação de aparelhos permite:
    - Cadastrar clientes e aparelhos.
    - Listar aparelhos disponíveis para locação.
    - Calcular o valor da locação de um aparelho por um determinado número de dias.
- Foram adicionados atributos e métodos que permite:
    - Listar os aparelhos alugados por um cliente.

```java
public class Cliente implements Serializable {
    private String nome;
    private String cpf;
    private List<Aparelho> aparelhosAlugados = new ArrayList<>();
```

    - Permitir que clientes aluguem um ou mais aparelhos.

```java
// Método para adicionar um aparelho alugado
public void alugarAparelho(Aparelho aparelho) {
    this.aparelhosAlugados.add(aparelho);
}
```

2. Classe LocacaoAplication:

- Classe principal (entry point) de uma aplicação Spring Boot. Ela é responsável por iniciar e configurar a aplicação.
- A anotação **@SpringBootApplication** é uma anotação de conveniência que combina três outras anotações:
- **@EnableAutoConfiguration** Habilita a configuração automática do Spring Boot. Isso significa que o Spring Boot tentará configurar automaticamente sua aplicação com base nas dependências que você adicionou ao projeto.
- **@ComponentScan** faz com que o Spring Boot escaneie o pacote atual e seus subpacotes em busca de componentes (como **@Controller**, **@Service**, **@Repository**, etc.) para serem gerenciados pelo Spring.
- **@configuration** permite registrar beans extras no contexto ou importar classes de configuração adicionais
- O método **SpringApplication.run** inicia a aplicação Spring Boot.

```java
@SpringBootApplication
public class LocacaoApplication {
    // Inicializa a aplicação
    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(LocacaoApplication.class, args);
    }
}
```

3. A classe LocacaoController:

- O controlador **LocacaoController** está anotado com **@RestControlle**r e mapeado para o caminho base **/locacao**, permitindo a exposição dos endpoints via HTTP.

```
@RestController
@RequestMapping("/locacao")
```

- **Endpoint: POST /locacao/clientes**
  - Recebe um objeto **Cliente** no corpo da requisição e o adiciona à lista de clientes.
  - Retorna uma mensagem de sucesso.
- **Endpoint: GET /locacao/clientes**
  - Retorna a lista de todos os clientes cadastrados.

```
// Cadastrar Cliente
@PostMapping("/clientes")
public String cadastrarCliente(@RequestBody Cliente cliente) {
    clientes.add(cliente);
    return "Cliente cadastrado com sucesso: " + cliente.getNome();
}

//Listar Clientes
@GetMapping("/clientes")
public List<Cliente> listarClientes() {
    return clientes;
}
```

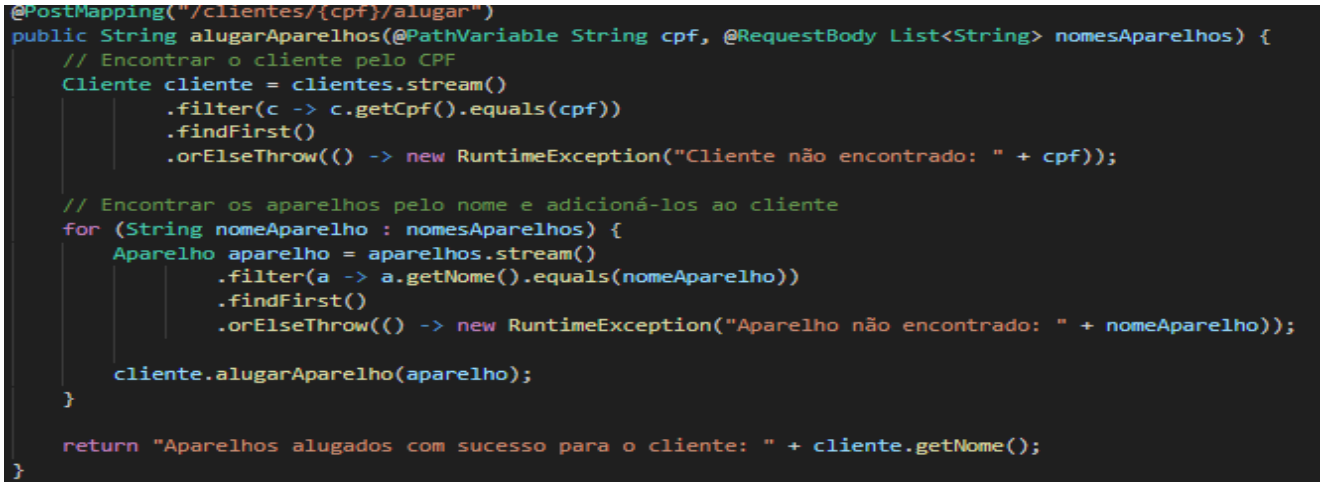
- **Endpoint: POST /locacao/aparelhos**
  - Recebe um objeto Aparelho no corpo da requisição e o adiciona à lista de aparelhos.
  - Retorna uma mensagem de sucesso.
- **Endpoint: GET /locacao/aparelhos**
  - Retorna a lista de todos os aparelhos cadastrados.

```
// Cadastrar Aparelho
@PostMapping("/aparelhos")
public String cadastrarAparelho(@RequestBody Aparelho aparelho) {
    aparelhos.add(aparelho);
    return "Aparelho cadastrado com sucesso: " + aparelho.getNome();
}

// Listar Aparelhos
@GetMapping("/aparelhos")
public List<Aparelho> listarAparelhos() {
    return aparelhos;
}
```

- **Endpoint: POST /locacao/clientes/{cpf}/alugar**

    - Recebe o CPF do cliente e uma lista de nomes de aparelhos no corpo da requisição.
    - Verifica se o cliente existe e se os aparelhos estão cadastrados.
    - Associa os aparelhos alugados ao cliente.
    - Retorna uma mensagem de sucesso.



```
@PostMapping("/clientes/{cpf}/alugar")
public String alugarAparelhos(@PathVariable String cpf, @RequestBody List<String> nomesAparelhos) {
    // Encontrar o cliente pelo CPF
    Cliente cliente = clientes.stream()
            .filter(c -> c.getCpf().equals(cpf))
            .findFirst()
            .orElseThrow(() -> new RuntimeException("Cliente não encontrado: " + cpf));

    // Encontrar os aparelhos pelo nome e adicioná-los ao cliente
    for (String nomeAparelho : nomesAparelhos) {
        Aparelho aparelho = aparelhos.stream()
                .filter(a -> a.getNome().equals(nomeAparelho))
                .findFirst()
                .orElseThrow(() -> new RuntimeException("Aparelho não encontrado: " + nomeAparelho));

        cliente.alugarAparelho(aparelho);
    }

    return "Aparelhos alugados com sucesso para o cliente: " + cliente.getNome();
}
```

- **Endpoint: GET /locacao/clientes/{cpf}/aparelhos**

    - Recebe o CPF do cliente como parâmetro.
    - Retorna a lista de aparelhos alugados por esse cliente.

- **Endpoint: DELETE /locacao/aparelhos/{nome}**
    - Recebe o nome do aparelho como parâmetro.
    - Procura o aparelho na lista e o remove.
    - Retorna uma mensagem de confirmação ou erro caso o aparelho não seja encontrado.

```
@GetMapping("/clientes/{cpf}/aparelhos")
public List<Aparelho> listarAparelhosAlugados(@PathVariable String cpf) {
    Cliente cliente = clientes.stream()
            .filter(c -> c.getCpf().equals(cpf))
            .findFirst()
            .orElseThrow(() -> new RuntimeException("Cliente não encontrado: " + cpf));

    return cliente.getAparelhosAlugados();
}

// Deletar Aparelho
@DeleteMapping("/aparelhos/{nome}")
public String deletarAparelho(@PathVariable String nome) {
    for (Aparelho aparelho : aparelhos) {
        if (aparelho.getNome().equals(nome)) {
            aparelhos.remove(aparelho);
            return "Aparelho deletado com sucesso: " + nome;
        }
    }
    return "Aparelho não encontrado: " + nome;
}
```

- **Endpoint: GET /locacao/calcularLocacao?nomeAparelho={nome}&dias={dias}**
    - Recebe o nome do aparelho e a quantidade de dias como parâmetros.
    - Calcula o valor total da locação com base no preço diário do aparelho.
    - Retorna o valor total ou uma mensagem de erro caso o aparelho não seja encontrado.

```java
// Calcular Locação
@GetMapping("/calcularLocacao")
public String calcularLocacao(@RequestParam String nomeAparelho, @RequestParam int dias) {
    for (Aparelho aparelho : aparelhos) {
        if (aparelho.getNome().equals(nomeAparelho)) {
            double valorTotal = aparelho.getPrecoLocacao() * dias;
            return "Locação do aparelho " + aparelho.getNome() + " por " + dias + " dias. Valor total: R$ "
                    + valorTotal;
        }
    }
    return "Aparelho não encontrado: " + nomeAparelho;
}
```

4. Testando as funcionalidades usando o Postman

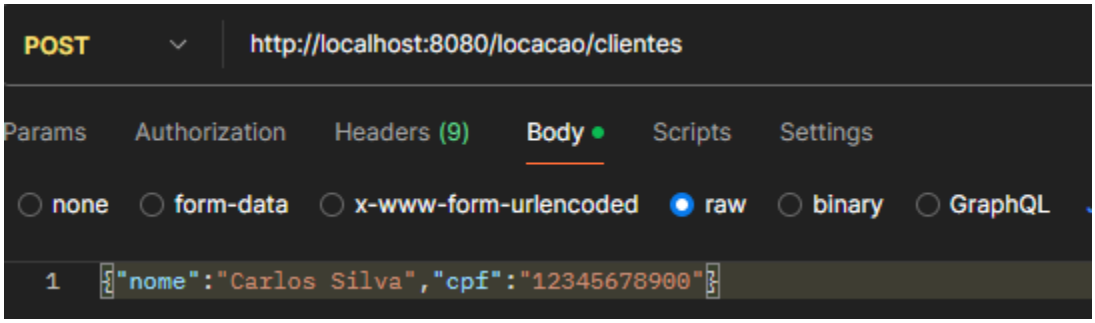
- **Iniciando o Spring Boot**

```
caiocezar@DESKTOP-0TV6PJ1:~/Trabalho-3-API$ mvn spring-boot:run
```

```
 .   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.0)

2025-03-02T13:15:20.660-03:00  INFO 72116 --- [           main] com.sd.locacao.LocacaoApplication        : Starting LocacaoApplication using Java 21.0.6 with PID 72116 (/home/caioc
r/Trabalho-3-API/target/classes started by caiocezar in /home/caiocezar/Trabalho-3-API)
2025-03-02T13:15:20.664-03:00  INFO 72116 --- [           main] com.sd.locacao.LocacaoApplication        : No active profile set, falling back to 1 default profile: "default"
2025-03-02T13:15:21.413-03:00  INFO 72116 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2025-03-02T13:15:21.422-03:00  INFO 72116 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-03-02T13:15:21.423-03:00  INFO 72116 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.8]
2025-03-02T13:15:21.526-03:00  INFO 72116 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-03-02T13:15:21.528-03:00  INFO 72116 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 819 ms
2025-03-02T13:15:21.802-03:00  INFO 72116 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2025-03-02T13:15:21.809-03:00  INFO 72116 --- [           main] com.sd.locacao.LocacaoApplication        : Started LocacaoApplication in 1.505 seconds (process running for 1.864)
2025-03-02T13:17:30.511-03:00  INFO 72116 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-03-02T13:17:30.512-03:00  INFO 72116 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-03-02T13:17:30.513-03:00  INFO 72116 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2025-03-02T13:20:51.563-03:00  WARN 72116 --- [nio-8080-exec-5] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.http.converter.HttpMessageNotReadableExcept
: JSON parse error: Unexpected character ('n' (code 110)): was expecting double-quote to start field name]
```

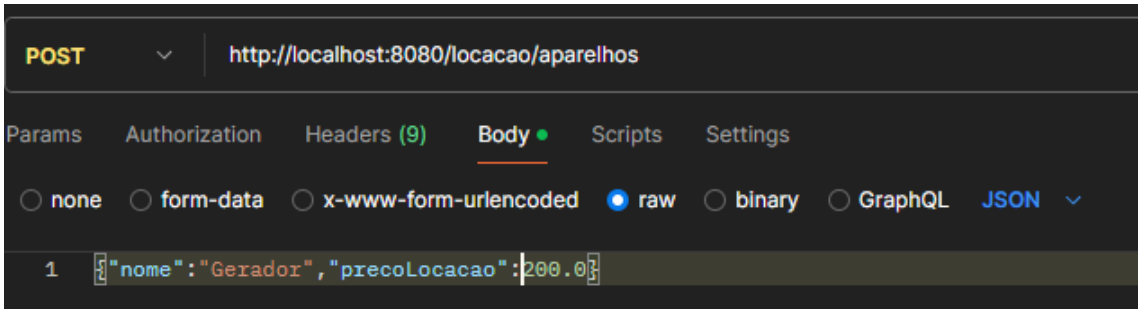
Cadastrar Clientes



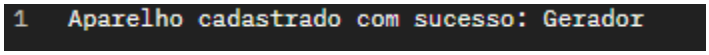
Resposta esperada

```
1   Cliente cadastrado com sucesso: Carlos Silva
```

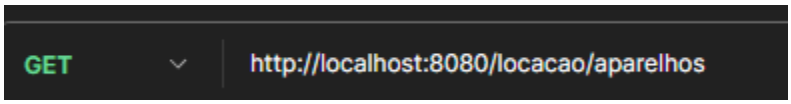
- **Cadastrar Aparelhos**



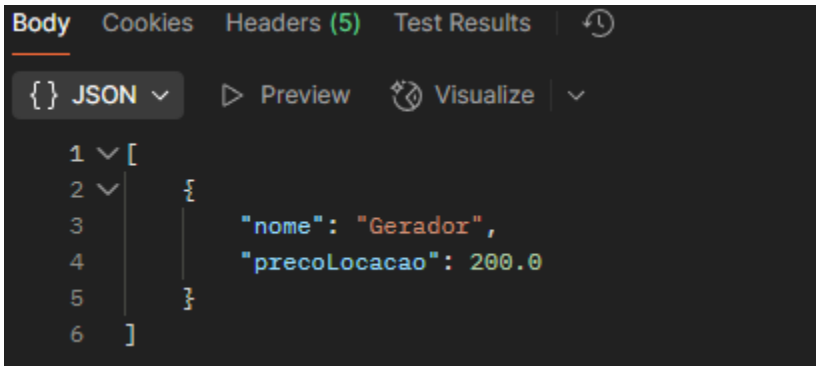
Resposta esperada



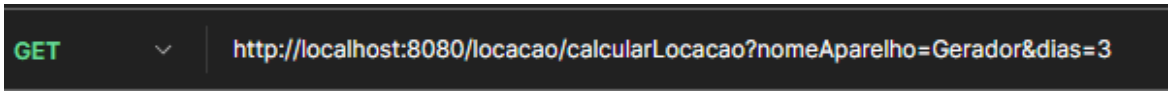
- **Listar Aparelhos Disponíveis**



Resposta esperada



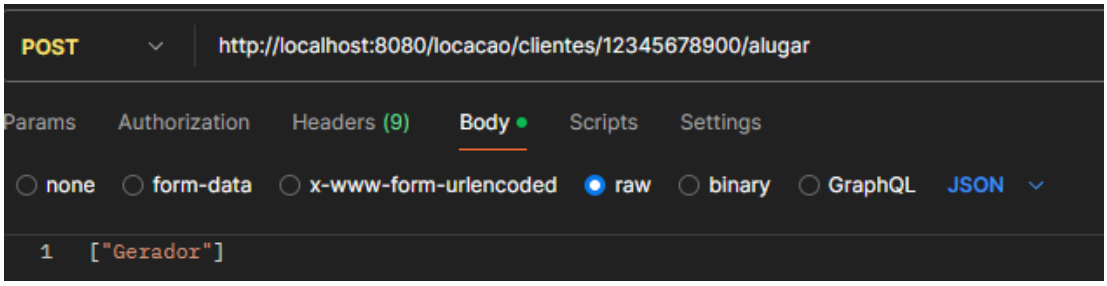
- **Calcular locação**



Resposta esperada

```
Locação do aparelho Gerador por 3 dias. Valor total: R$ 600.0
```

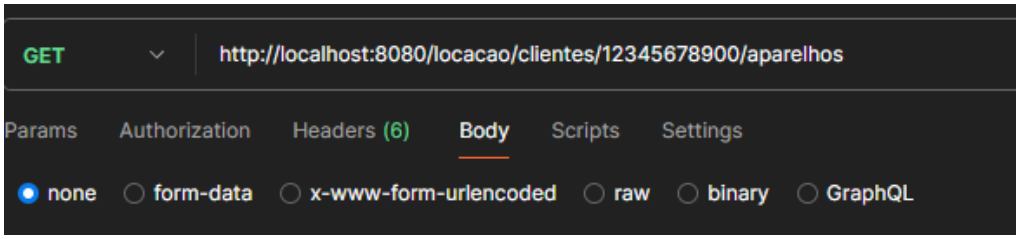
- **Alugar aparelho**



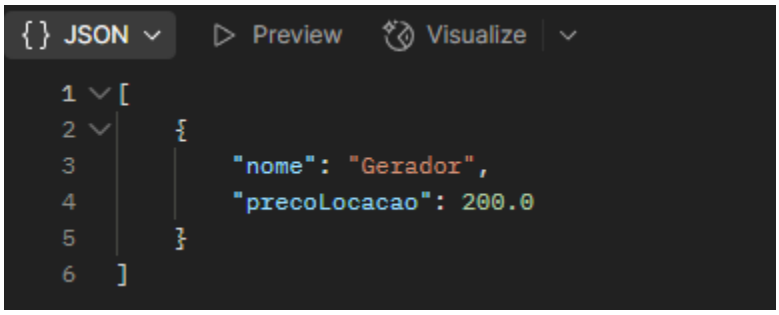
Resposta esperada

```
Aparelhos alugados com sucesso para o cliente: Carlos Silva
```

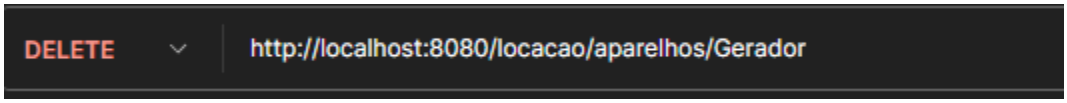
- **Listar aparelhos alugados por cliente**



Resposta esperada



- **Deletar aparelhos**



Resposta esperada

```
1   Aparelho deletado com sucesso: Gerador
```