

DOCUMENTAÇÃO DA API DE PESQUISA DO UBURU

1. Mecanismo de pesquisa

1.1. Comunicação com a API

O endereço da API do mecanismo de pesquisa será “localhost:8080/api/v1/search” e os métodos HTTP podem ser GET ou POST. Para se comunicar com a API, será necessário inserir, no mínimo, os seguintes dados:

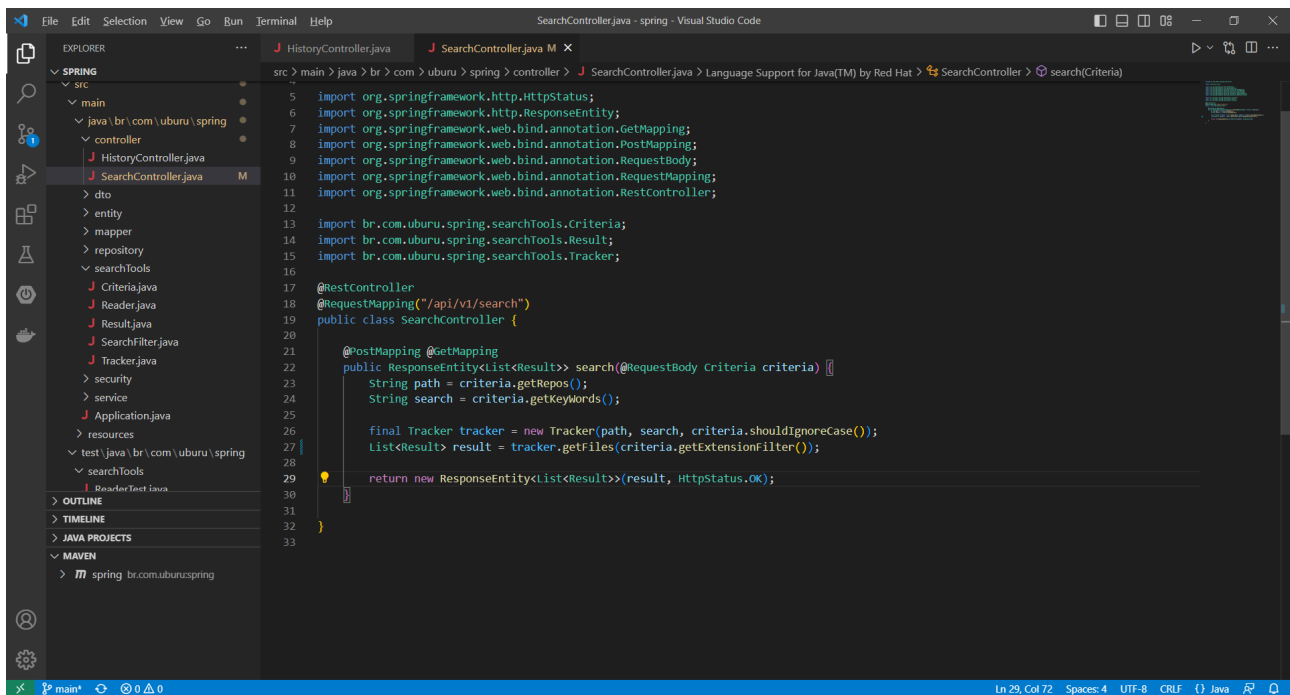
```
public class Criteria {  
  
    private String keyWords;  
    private String repos;  
    private String extensionFilter;  
    private boolean ignoreCase;  
  
    public String getKeyWords() {  
        return keyWords;  
    }  
  
    public void setKeyWords(String keyWords) {  
        this.keyWords = keyWords;  
    }  
  
    public String getRepos() {  
        return repos;  
    }  
}
```

- keyWords: As palavras a serem pesquisadas
- repos: Diretórios ou arquivos onde deve ser realizada a pesquisa
- extensionFilter: Extensões de arquivos ACEITAS
- ignoreCase: Se deve considerar a capitalização dos caracteres

O retorno será um JSON contendo as linhas, seu texto e o nome do arquivo em que foram encontradas as palavras.

1.2. Funcionamento do backend

A classe SearchController.java é o endpoint que recebe a requisição. Ela instanciará a classe Tracker.java, passando os critérios de pesquisa no construtor.

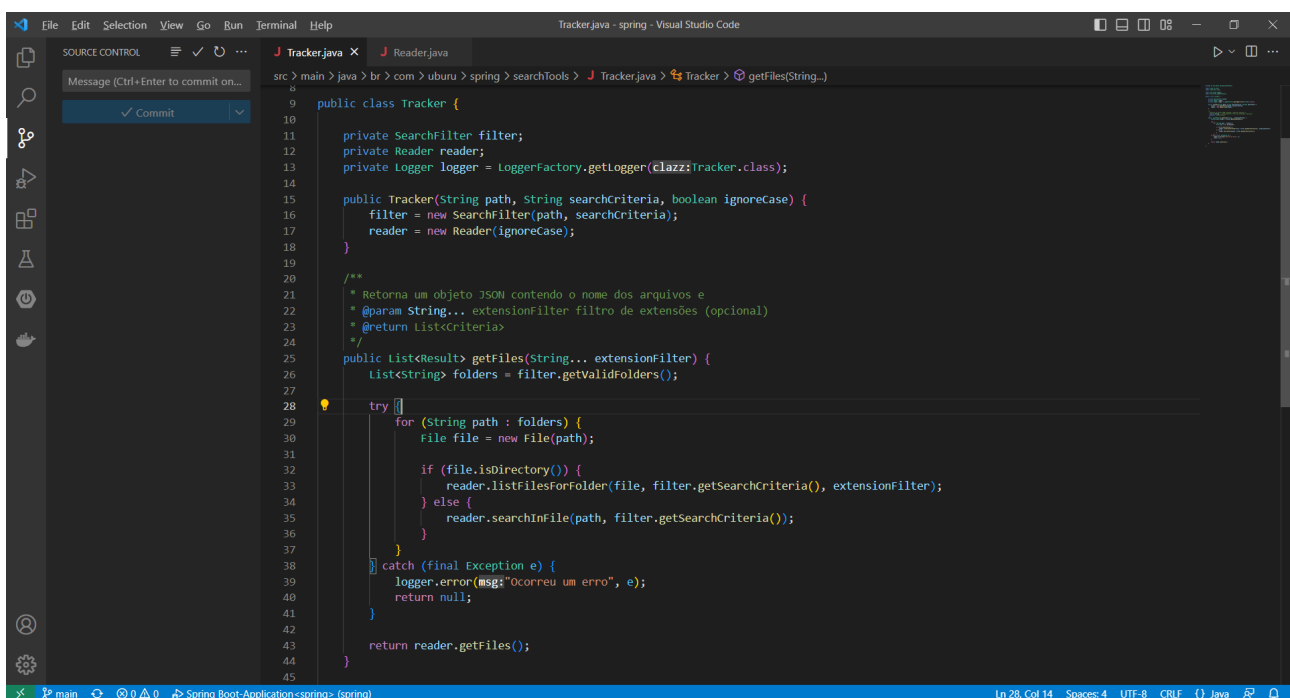


```
5 import org.springframework.http.HttpStatus;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PostMapping;
9 import org.springframework.web.bind.annotation.RequestBody;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import br.com.uburu.spring.searchTools.Criteria;
14 import br.com.uburu.spring.searchTools.Result;
15 import br.com.uburu.spring.searchTools.Tracker;
16
17 @RestController
18 @RequestMapping("/api/v1/search")
19 public class SearchController {
20
21     @PostMapping @GetMapping
22     public ResponseEntity<List<Result>> search(@RequestBody Criteria criteria) {
23         String path = criteria.getRepos();
24         String search = criteria.getKeywords();
25
26         final Tracker tracker = new Tracker(path, search, criteria.shouldIgnoreCase());
27         List<Result> result = tracker.getFiles(criteria.getExtensionFilter());
28
29         return new ResponseEntity<List<Result>>(result, HttpStatus.OK);
30     }
31 }
32
33 }
```

Para definir quais são os critérios de pesquisa, foi criada uma classe chamada SearchFilter.java, que contém os métodos que separam as strings, por exemplos, dos paths de pesquisa, em um Array de Strings e faz a verificação de quais paths são válidos e inválidos.

OBS. 1: Caminhos ou paths inválidos são aqueles que possuem uma exclamação (!) na frente

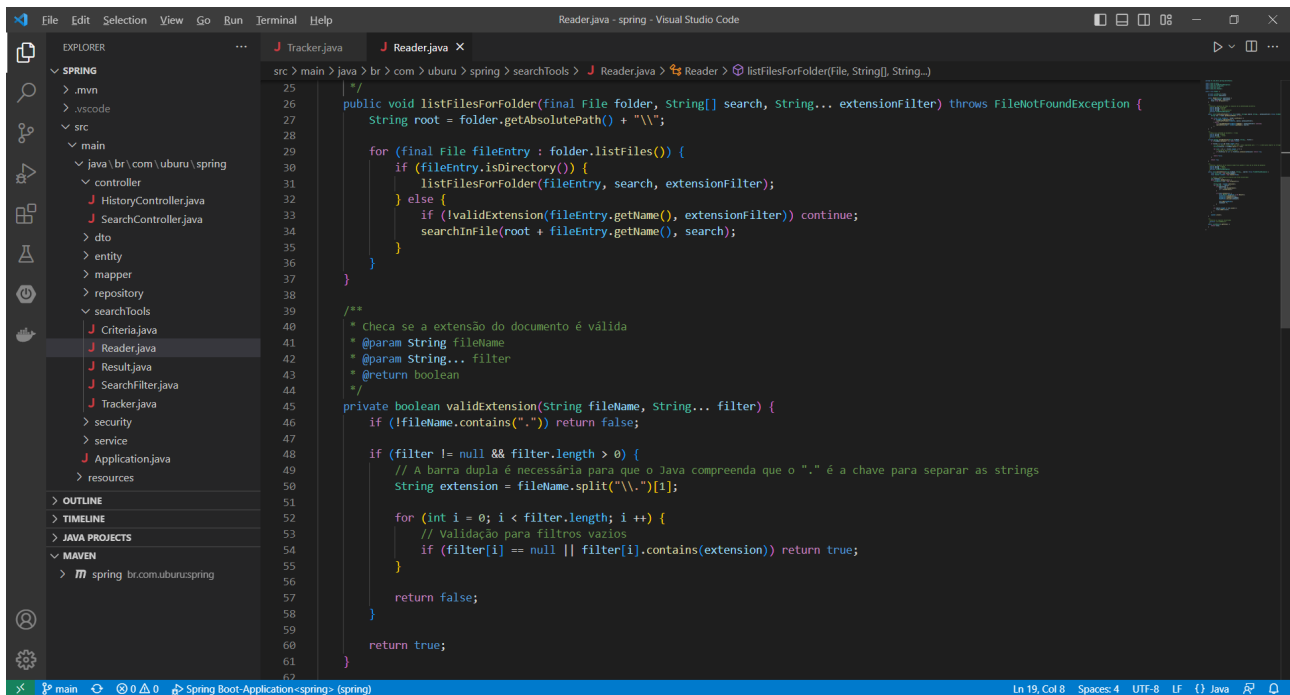
Na classe Tracker.java, o método getFiles() decidirá entre dois métodos distintos (listFilesForFolder() e searchInFile()). A diferença entre eles é justamente que um dos métodos pretende realizar a pesquisa em um diretório, enquanto o outro pretende fazer isso em apenas um arquivo.



```
9 public class Tracker {
10
11     private SearchFilter filter;
12     private Reader reader;
13     private Logger logger = LoggerFactory.getLogger(clazz::Tracker.class);
14
15     public Tracker(String path, String searchCriteria, boolean ignoreCase) {
16         filter = new SearchFilter(path, searchCriteria);
17         reader = new Reader(ignoreCase);
18     }
19
20     /**
21      * Retorna um objeto JSON contendo o nome dos arquivos e
22      * @param String... extensionFilter filtro de extensões (opcional)
23      * @return List<Criteria>
24      */
25     public List<Result> getFiles(String... extensionFilter) {
26         List<String> folders = filter.getValidFolders();
27
28         try {
29             for (String path : folders) {
30                 File file = new File(path);
31
32                 if (file.isDirectory()) {
33                     reader.listFilesForFolder(file, filter.getSearchCriteria(), extensionFilter);
34                 } else {
35                     reader.searchInFile(path, filter.getSearchCriteria());
36                 }
37             }
38         } catch (final Exception e) {
39             logger.error(msg:"Ocorreu um erro", e);
40             return null;
41         }
42
43         return reader.getFiles();
44     }
45 }
```

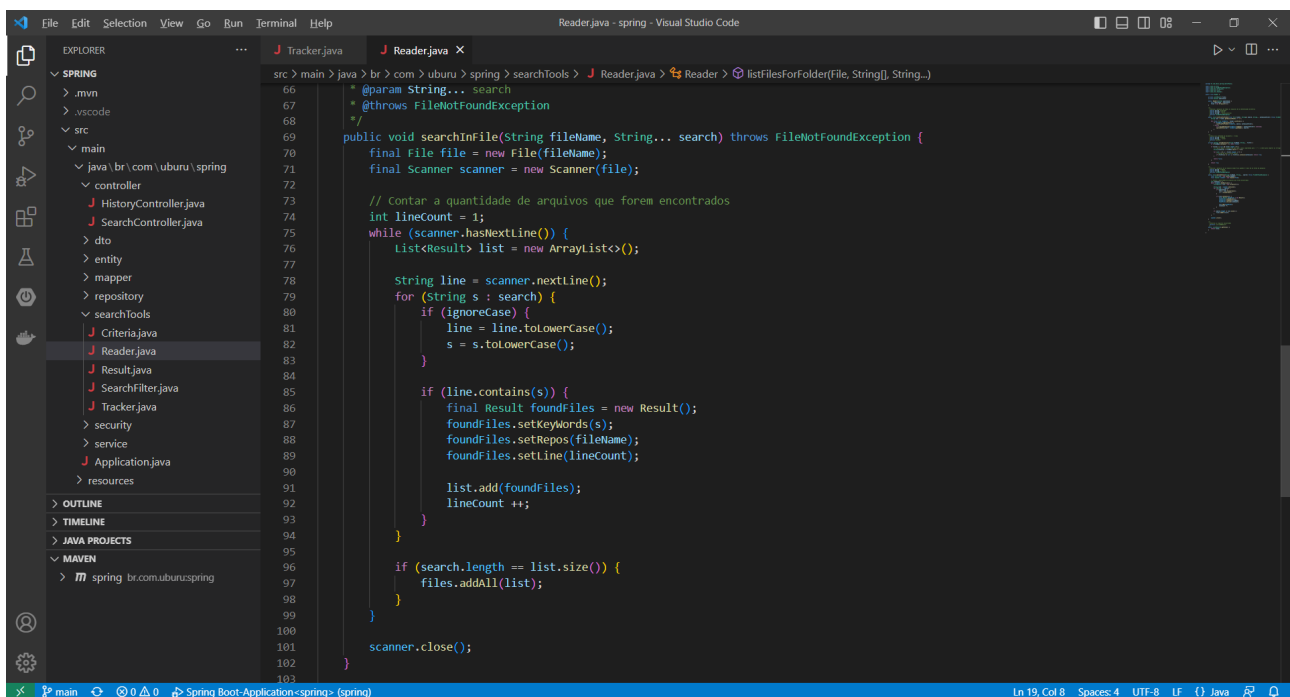
Para finalizar, na classe Reader.java, teremos os métodos que analisam os arquivos individualmente. O principal deles é o método searchInFile, que realiza a busca das palavras em um

determinado arquivo. O método supracitado como uma das “escolhas” da classe Tracker.java – o método – será responsável apenas por verificar as extensões dos arquivos, validando-os e “navegar” pelos diretórios, chamando o método searchInFile em cada arquivo válido.



```
src > main > java > br > com > uburu > spring > searchTools > J Reader.java > Reader > listFilesForFolder(File, String[], String...)

25  */
26  public void listFilesForFolder(final File folder, String[] search, String... extensionFilter) throws FileNotFoundException {
27      String root = folder.getAbsolutePath() + "\\";
28
29      for (final File fileEntry : folder.listFiles()) {
30          if (fileEntry.isDirectory()) {
31              listFilesForFolder(fileEntry, search, extensionFilter);
32          } else {
33              if (lvalidExtension(fileEntry.getName(), extensionFilter)) continue;
34              searchInFile(root + fileEntry.getName(), search);
35          }
36      }
37  }
38
39  /**
40   * Checa se a extensão do documento é válida
41   * @param String fileName
42   * @param String... filter
43   * @return boolean
44   */
45  private boolean validExtension(String fileName, String... filter) {
46      if (!fileName.contains(".")) return false;
47
48      if (filter != null && filter.length > 0) {
49          // A barra dupla é necessária para que o Java compreenda que o "." é a chave para separar as strings
50          String extension = fileName.split("\\.")[1];
51
52          for (int i = 0; i < filter.length; i++) {
53              // Validação para filtros vazios
54              if (filter[i] == null || filter[i].contains(extension)) return true;
55          }
56
57          return false;
58      }
59
60      return true;
61  }
62  }
```



```
src > main > java > br > com > uburu > spring > searchTools > J Reader.java > Reader > listFilesForFolder(File, String[], String...)

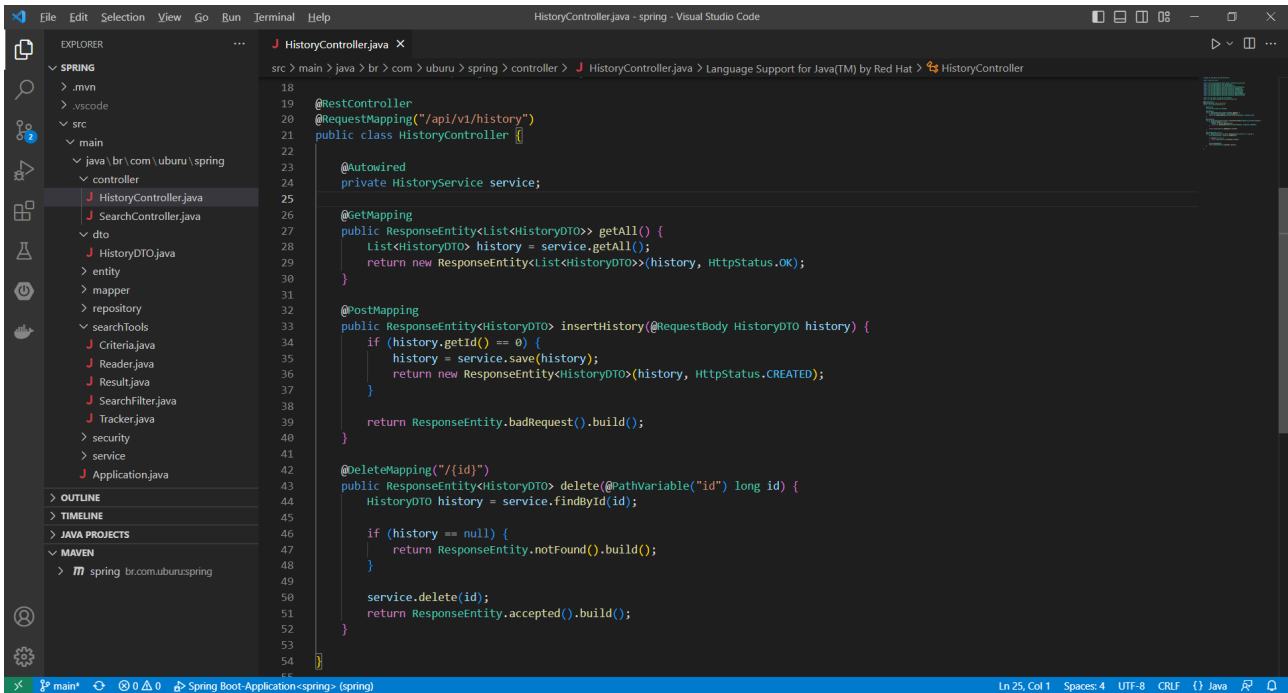
66  * @param String... search
67  * @throws FileNotFoundException
68  */
69  public void searchInFile(String fileName, String... search) throws FileNotFoundException {
70      final File file = new File(fileName);
71      final Scanner scanner = new Scanner(file);
72
73      // Contar a quantidade de arquivos que forem encontrados
74      int lineCount = 1;
75      while (scanner.hasNextLine()) {
76          List<Result> list = new ArrayList<>();
77
78          String line = scanner.nextLine();
79          for (String s : search) {
80              if (ignoreCase) {
81                  line = line.toLowerCase();
82                  s = s.toLowerCase();
83              }
84
85              if (line.contains(s)) {
86                  final Result foundFiles = new Result();
87                  foundFiles.setKeywords(s);
88                  foundFiles.setRepos(fileName);
89                  foundFiles.setLine(lineCount);
90
91                  list.add(foundFiles);
92                  lineCount++;
93              }
94          }
95
96          if (search.length == list.size()) {
97              files.addAll(list);
98          }
99      }
100
101      scanner.close();
102  }
103  }
```

OBS. 2: A API do backend não verificará se o mesmo arquivo já apareceu nos resultados e uma mesma linha pode aparecer múltiplas vezes. Isso será responsabilidade do frontend.

2. Histórico

2.1. Registro de histórico de pesquisa

O endereço da API do mecanismo de pesquisa será “localhost:8080/api/v1/history” e os métodos HTTP podem ser GET, POST ou DELETE – cada um para realização de uma determinada operação.



- O método GET busca todos os registros do histórico e não recebe nenhum parâmetro
- O método POST inserirá um novo registro no histórico e deve receber como parâmetro a data do registro (formato yyyy-MM-dd), a string com os caminhos selecionados e a string das palavras pesquisadas.
- O método DELETE excluirá um registro de histórico do banco de dados.

2.2. Funcionamento da API do histórico

A API do histórico é composta pelos componentes padrão de uma API SpringBoot comum, contendo uma entidade da tabela de histórico e sua correspondente classe DTO, um repository, um mapper, o service e sua implementação.