

## CRUD de Biblioteca

Igor Campos de Borba

Cadastro de livros, com login e consumo de api de terceiros. Implementado Front-end em React, Back-end em Spring Boot e Banco de dados H2 de testes.

**GitHub:** <https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2>

Guia para executar:

- Na pasta do front-end execute:

npm install para instalar dependências

npm start para executar a aplicação e acessar <http://localhost:3000>

- Na pasta do back-end execute:

com o IntelliJ aberto, execute a classe Library.java com botão direito run

- No postman, importe o arquivo e execute os endpoints:

Arquivo: [Postman - Library com login.postman\\_collection.json](#)

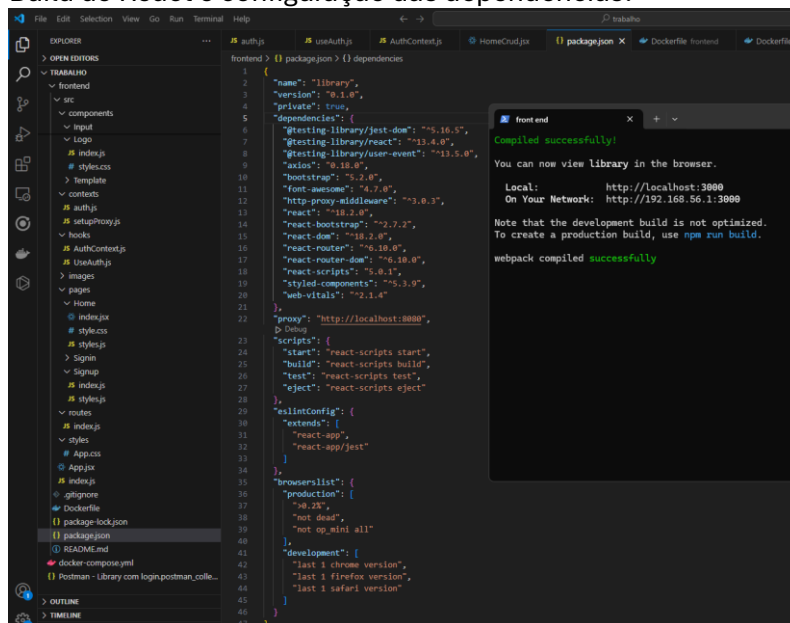
- Testes: no spring boot, na pasta principal clique em botão direito execute all tests.

Desafios encontrados: Login com OAuth2 e JWT, deploy do Docker.

### 0 Integração do Frontend e Backend

- Configuração do ambiente React e Spring Boot
- Controller no back-end e front-end
- Layout

Build do React e configuração das dependências:



```
1 {
2   "name": "library",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^5.16.5",
7     "@testing-library/react": "^13.4.0",
8     "@testing-library/user-event": "^13.5.0",
9     "axios": "9.18.4",
10    "bootstrap": "5.2.0",
11    "font-awesome": "4.7.0",
12    "http-proxy-middleware": "3.0.3",
13    "react": "18.2.0",
14    "react-bootstrap": "2.7.2",
15    "react-dom": "18.2.0",
16    "react-router": "6.18.0",
17    "react-router-dom": "6.18.0",
18    "react-scripts": "5.0.1",
19    "styled-components": "5.3.9",
20    "web-vitals": "2.1.4",
21  },
22  "proxy": "http://localhost:8080",
23  "scripts": {
24    "start": "react-scripts start",
25    "build": "react-scripts build",
26    "test": "react-scripts test",
27    "eject": "react-scripts eject",
28  },
29  "eslintConfig": {
30    "extends": [
31      "react-app",
32      "react-app/jest"
33    ]
34  },
35  "browserslist": {
36    "production": [
37      ">0.2%",
38      "not dead",
39      "not op_mini all"
40    ],
41    "development": [
42      "last 1 chrome version",
43      "last 1 firefox version",
44      "last 1 safari version"
45    ]
46  }
47 }
```

front end

Compiled successfully!

You can now view library in the browser.

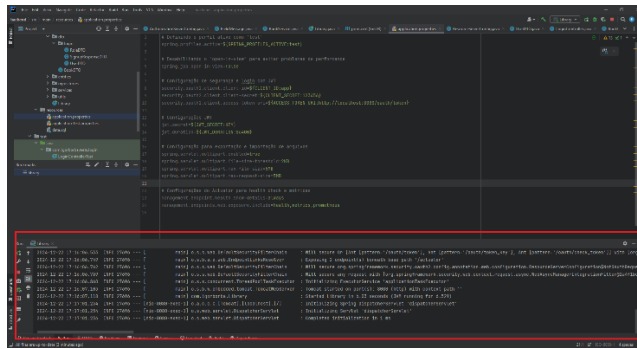
Local: <http://localhost:3000>  
On Your Network: <http://192.168.56.1:3000>

Note that the development build is not optimized.  
To create a production build, use `npm run build`.

webpack compiled successfully

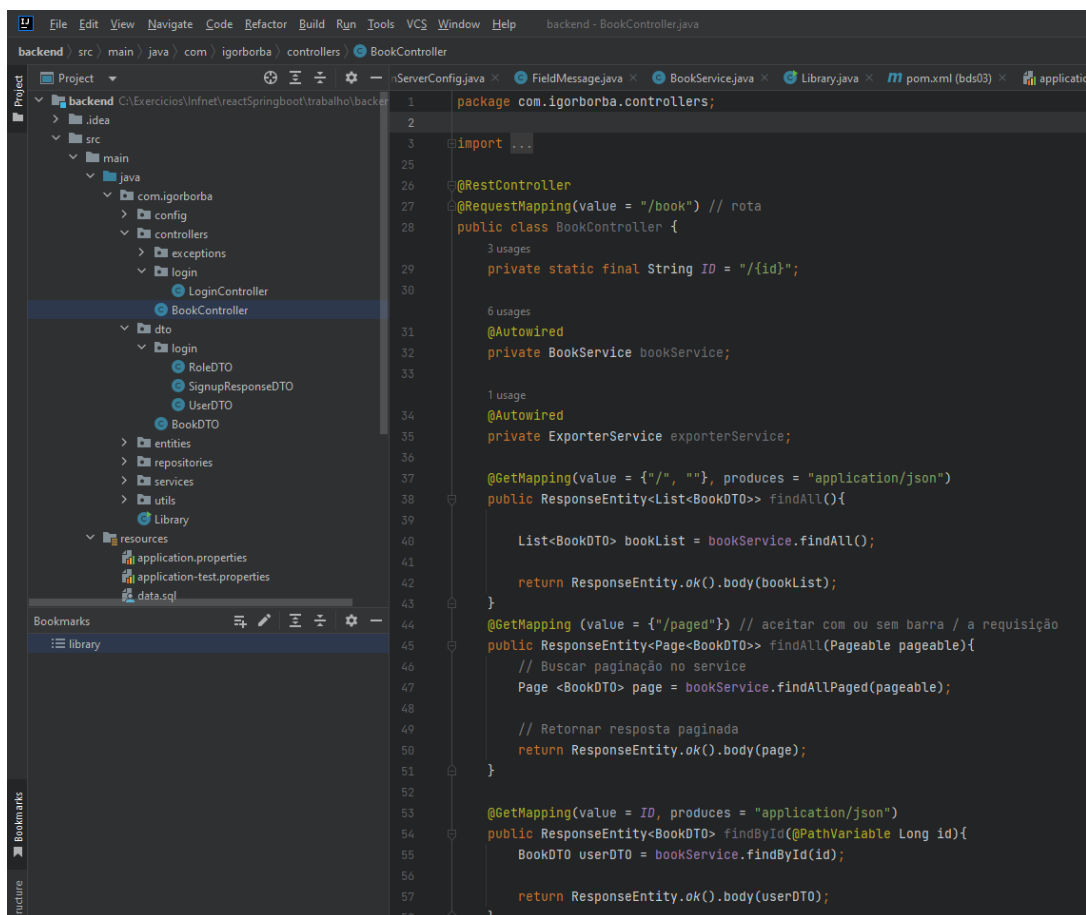
<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/package.json>

Build do Spring Boot, configuração do projeto e das dependências:



Configuração do projeto: <https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/resources/application.properties>

Dependências: <https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/pom.xml> Controller configurado no Back-end para tabela de livros:



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/BookController.java>

## Controller configurado no Front-end para tabela de livros:

<https://github.com/igorcamosdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/components/HomeCrud/HomeCrud.jsx>

Layout: Design responsivo para computador, tablet e smartphone:

localhost:3000/book

Home

Cadastro de livros

Cadastro

Livros populares

Exportar Livros

## Cadastro de Livros

Título do livro  
 Digite o título do livro

Nome do autor  
 Nome do autor

Sobrenome  
 Sobrenome

Editora  
 Editora

Edição  
 Ano

Edição  
 Ano

Cancelar Salvar

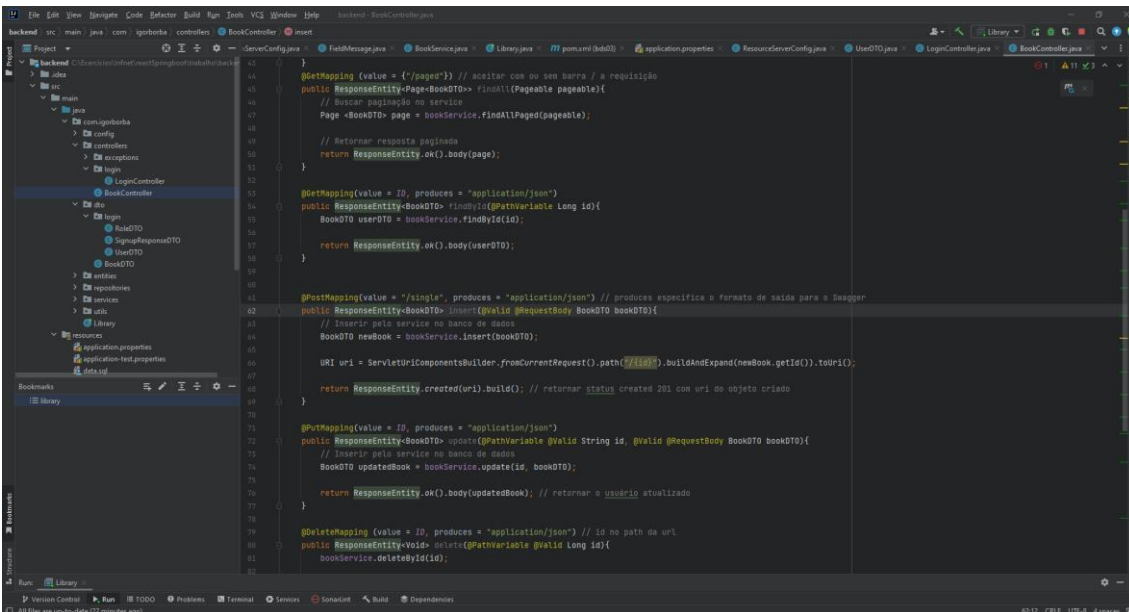
Título	Nome do autor	Sobrenome	Editora	Edição	Ano	Ações
Java para iniciantes: crie, compile e execute programas Java rapidamente	Herbert	SCHILD	Sapientza	3	2014	✏️ 🗑️
Java: como programar	Paul	DEITEL	Person	10	2020	✏️ 🗑️

Desenvolvido com ❤️ por Igor Borba

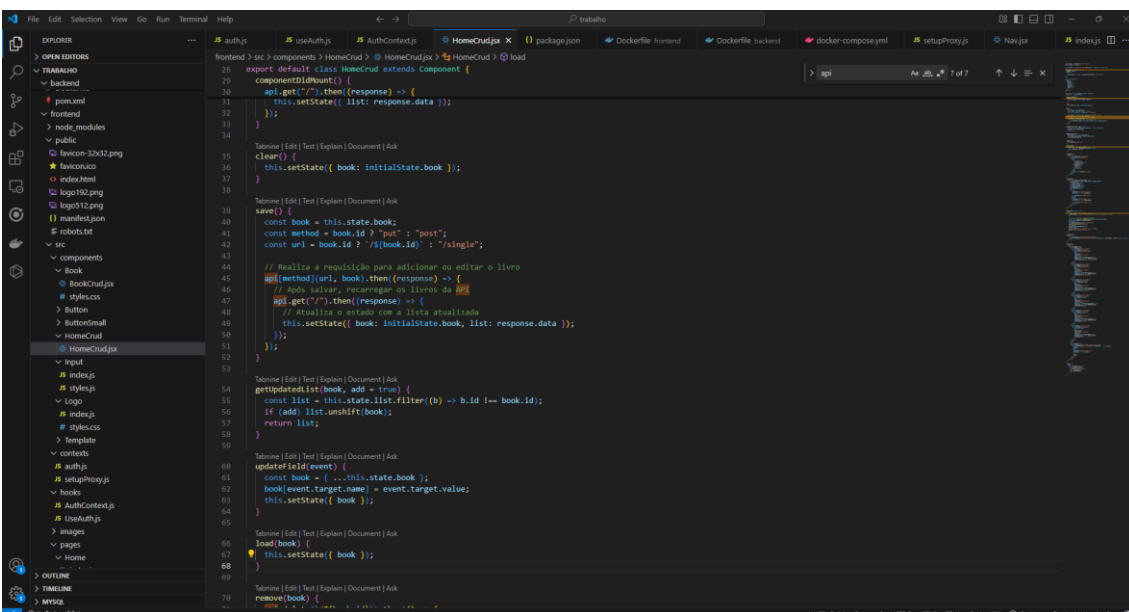
## 1

- Correta comunicação entre front-end e back-end;
- Validações e exceptions.

Comunicação via controller implementado com padrão REST API.

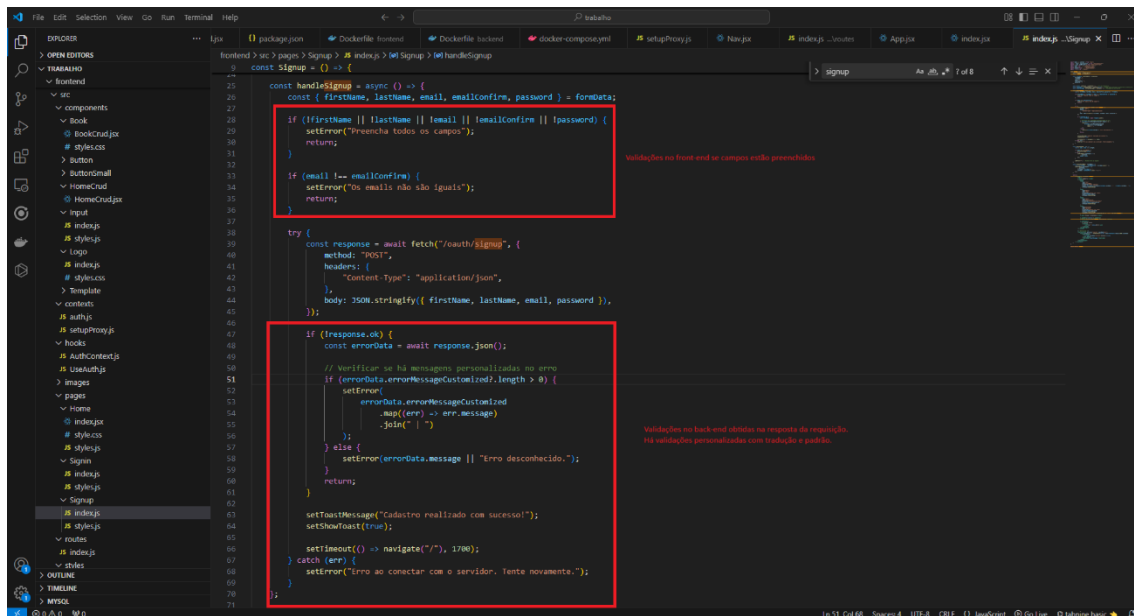


<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/BookController.java>



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/components/HomeCrud/HomeCrud.jsx>

Validações no front-end e no back-end. Validação no front-end evita consumo de recursos do back-end.



```
const handleSignup = async () => {
  const { firstName, lastName, email, emailConfirm, password } = formData;

  if ((firstName || lastName || email || emailConfirm || password) && !password) {
    setError("Preencha todos os campos");
    return;
  }

  if (email !== emailConfirm) {
    setError("Os emails não são iguais");
    return;
  }

  try {
    const response = await fetch("/auth/signup", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ firstName, lastName, email, password }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      // Verificar se há mensagens personalizadas no erro
      if (errorData.errorMessageCustomized.length > 0) {
        setError(
          errorData.errorMessageCustomized
            .map((err) => err.message)
            .join(" ")
        );
      } else {
        setError(errorData.message || "Erro desconhecido.");
      }
      return;
    }

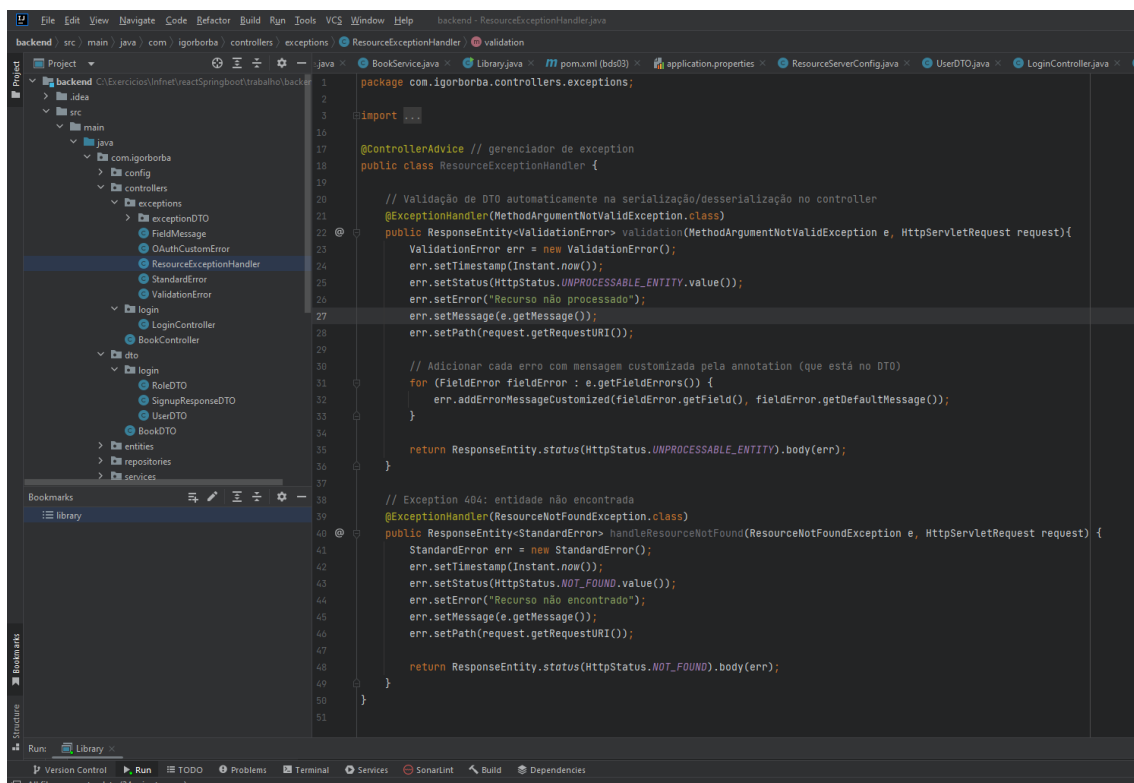
    showToastMessage("Cadastro realizado com sucesso!");
    setShowToast(true);
    setTimeout(() => navigate("/"), 1700);
  } catch (err) {
    setError("Erro ao conectar com o servidor. Tente novamente.");
  }
}
```

Validações no front-end se campos estão preenchidos

Validações no back-end obtidas na resposta de requisição. Há validações personalizadas com tradução e padrão.

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/pages/Signup/index.js>

Validações no back-end traduzidas, com handler de exceções personalizadas.



```
package com.igorborba.controllers.exceptions;

import ...

@ControllerAdvice // gerenciador de exception
public class ResourceExceptionHandler {

  // Validação de DTO automaticamente na serialização/desserialização no controller
  @ExceptionHandler(MethodArgumentNotValidException.class)
  public ResponseEntity<ValidationErrorResponse> validation(MethodArgumentNotValidException e, HttpServletRequest request) {
    ValidationErrorResponse err = new ValidationErrorResponse();
    err.setTimestamp(Instant.now());
    err.setStatus(HttpStatus.UNPROCESSABLE_ENTITY.value());
    err.setError("Recurso não processado");
    err.setMessage(e.getMessage());
    err.setPath(request.getRequestURI());

    // Adicionar cada erro com mensagem customizada pela annotation (que está no DTO)
    for (FieldError fieldError : e.getFieldErrors()) {
      err.addErrorMessageCustomized(fieldError.getField(), fieldError.getDefaultMessage());
    }

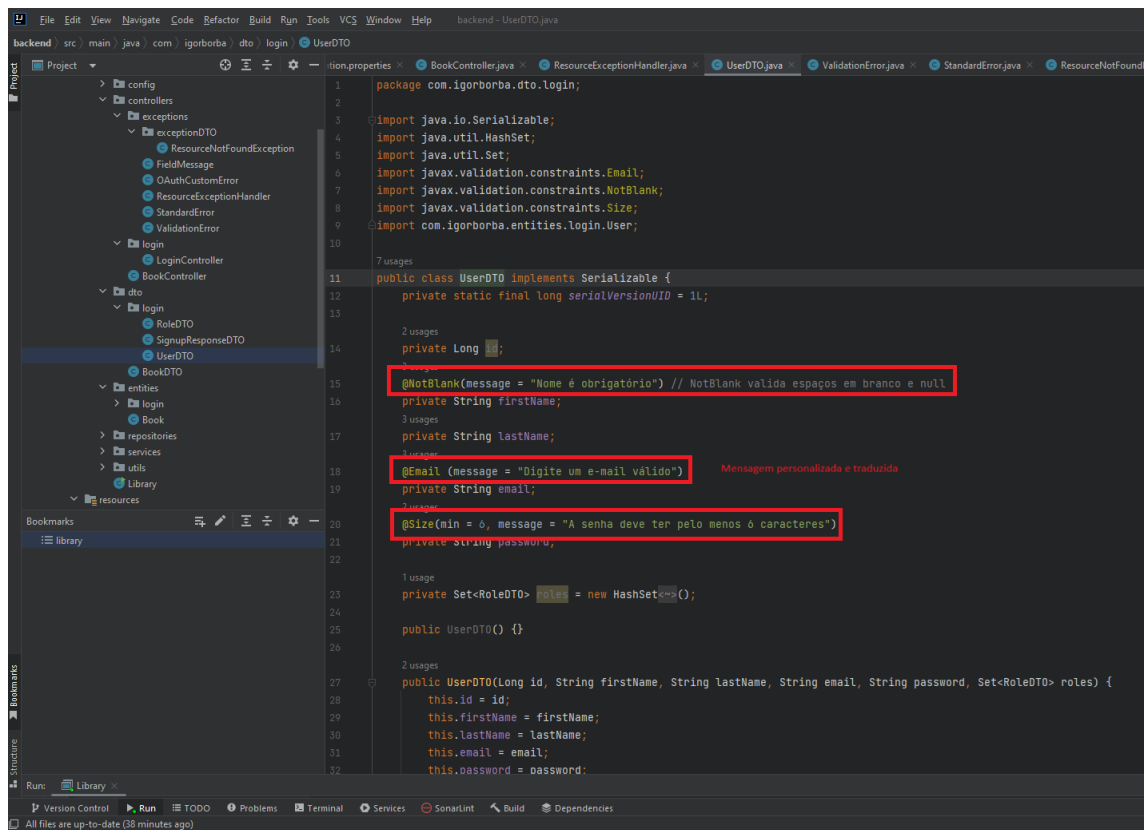
    return ResponseEntity.status(HttpStatus.UNPROCESSABLE_ENTITY).body(err);
  }

  // Exception 404: entidade não encontrada
  @ExceptionHandler(ResourceNotFoundException.class)
  public ResponseEntity<StandardError> handleResourceNotFound(ResourceNotFoundException e, HttpServletRequest request) {
    StandardError err = new StandardError();
    err.setTimestamp(Instant.now());
    err.setStatus(HttpStatus.NOT_FOUND.value());
    err.setError("Recurso não encontrado");
    err.setMessage(e.getMessage());
    err.setPath(request.getRequestURI());

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(err);
  }
}
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/exceptions/ResourceExceptionHandler.java>

Validações no back-end traduzidas, com handler de exceções personalizadas.



The screenshot shows an IDE window with the file `UserDTO.java` open. The code defines a `UserDTO` class that implements `Serializable`. It includes several validation annotations on its fields, which are highlighted with red boxes in the image:

- `@NotBlank(message = "Nome é obrigatório")` on the `firstName` field. A tooltip shows the message: "NotBlank valida espaços em branco e null".
- `@Email(message = "Digite um e-mail válido")` on the `email` field. A tooltip shows the message: "Mensagem personalizada e traduzida".
- `@Size(min = 6, message = "A senha deve ter pelo menos 6 caracteres")` on the `password` field.

The code also includes imports for `java.io.Serializable`, `java.util.HashSet`, `java.util.Set`, `javax.validation.constraints.Email`, `javax.validation.constraints.NotBlank`, `javax.validation.constraints.Size`, and `com.igorborba.entities.Login.User`. The `UserDTO` class has a constructor and a setter for the `roles` field.

<https://github.com/igorcamosdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/dto/login/UserDTO.java>

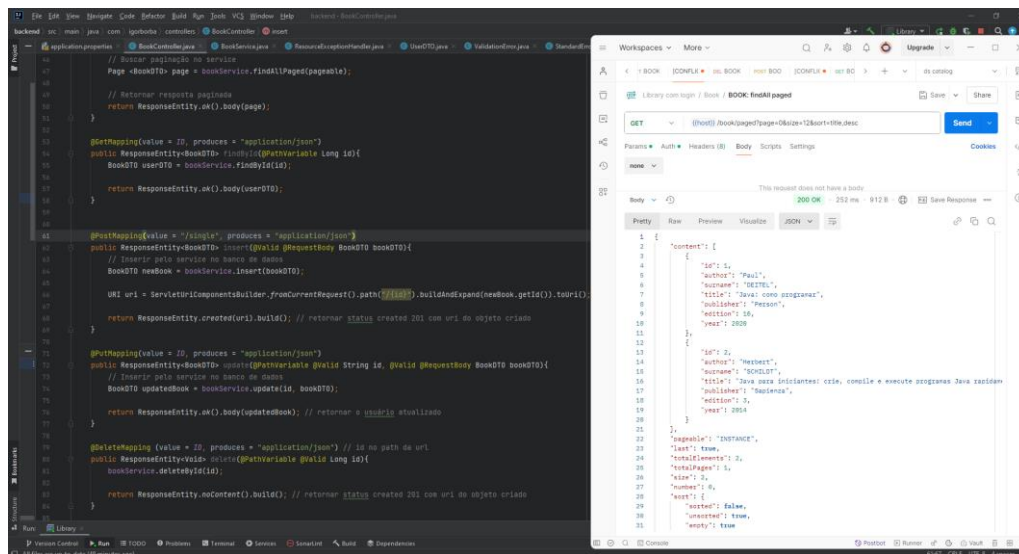
## 2 Criando e Consultando Dados com CRUD

- Read e Create funcionando.

FindAll implementado com e sem paginação, insert com validação por ser uma alteração não idempotente, já que vai alterar o registro no banco de dados. A paginação possui personalização de itens para buscar e ordem ascendente ou descendente conforme o título do livro.

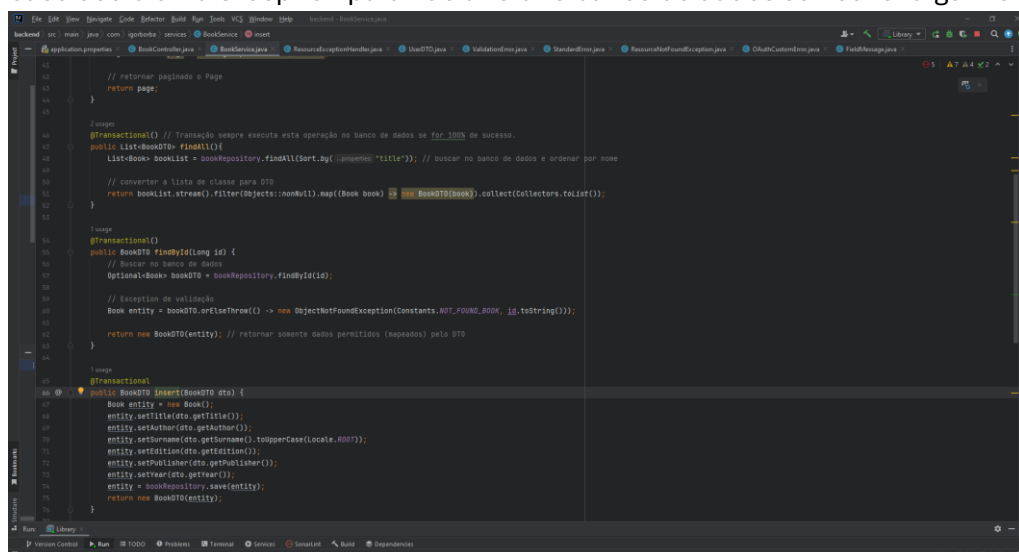
Há a separação de responsabilidades em camadas: no back-end, o controller recebe a requisição, usa um DTO para fins de segurança e delega a lógica da aplicação o service, que acessa o banco de dados.

Controller com findAll e insert de livros:

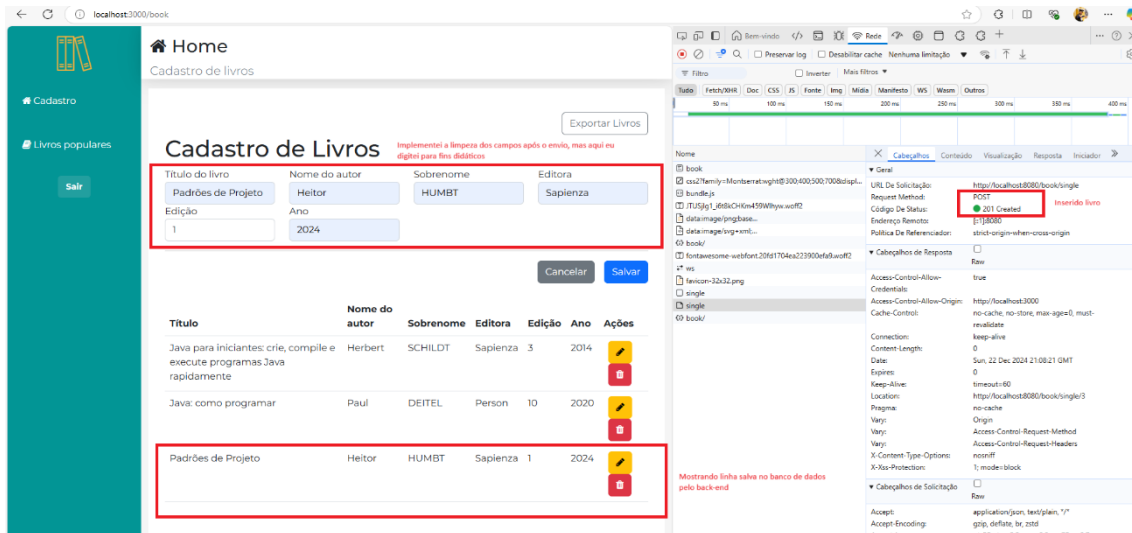


<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/BookController.java>

Service com acesso ao banco de dados, validações e Transações para possível rollback caso ocorra uma exception para não alterar o banco de dados se houver algum erro.



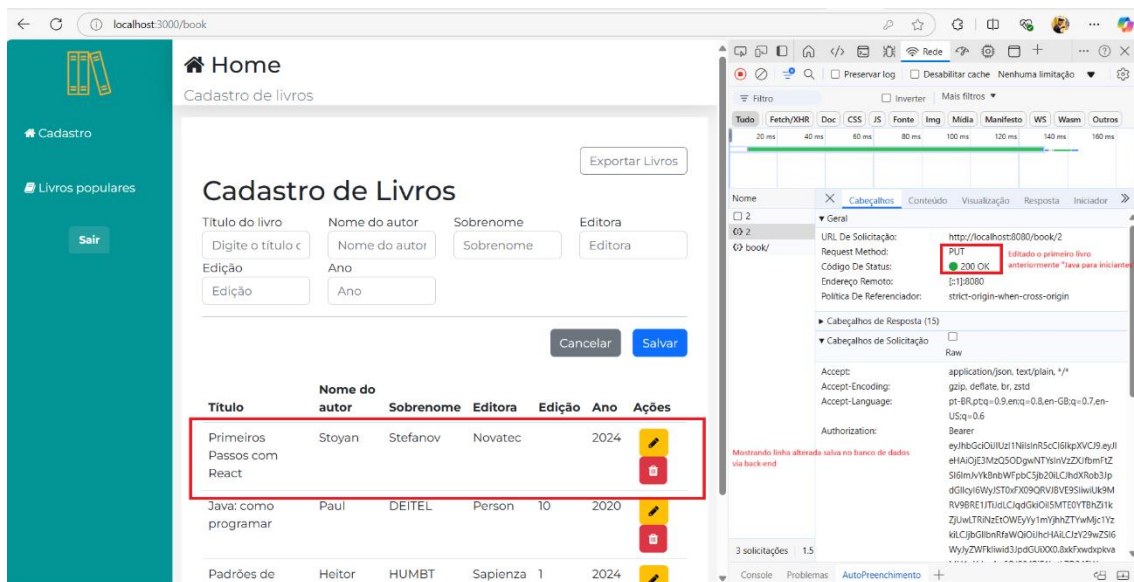
<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/services/BookService.java>



### 3 Atualizando e Excluindo Dados com CRUD

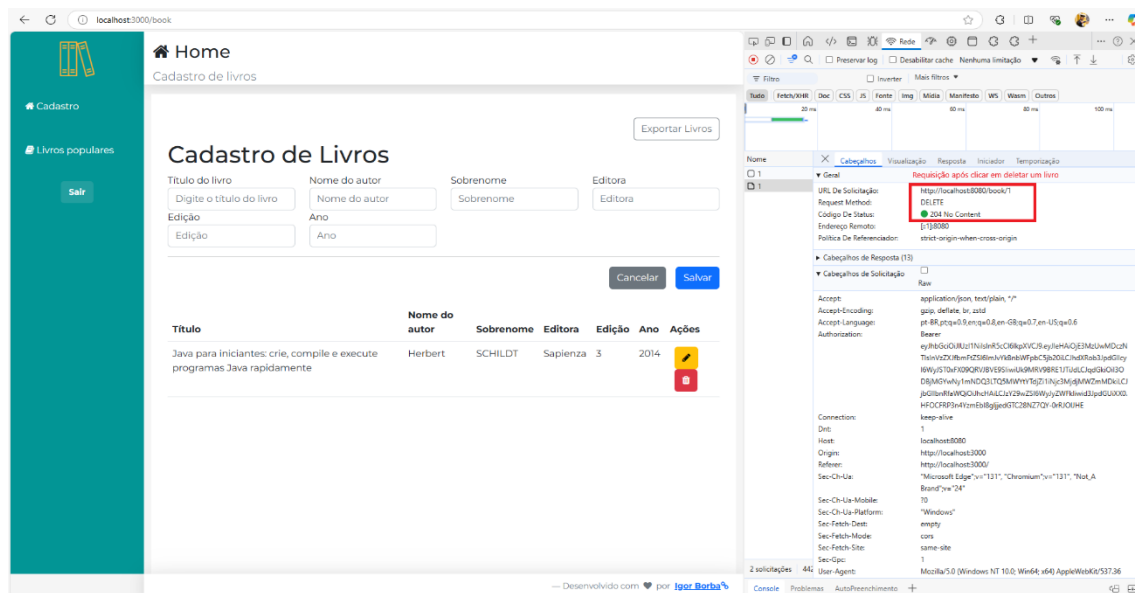
- Update e Delete funcionando;
- Validações e exceptions.

Update com validação por ser uma alteração não idempotente, já que vai alterar o registro no banco de dados. DeleteById possui validação no service do back-end se existe o recurso e retorna uma mensagem traduzida.

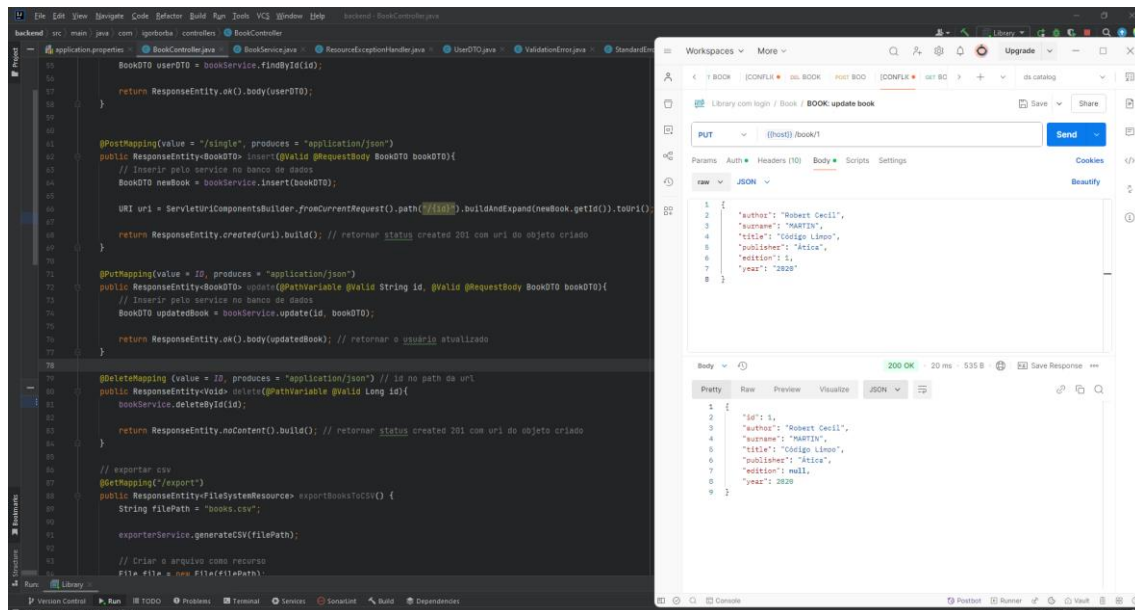




Delete por id do livro:

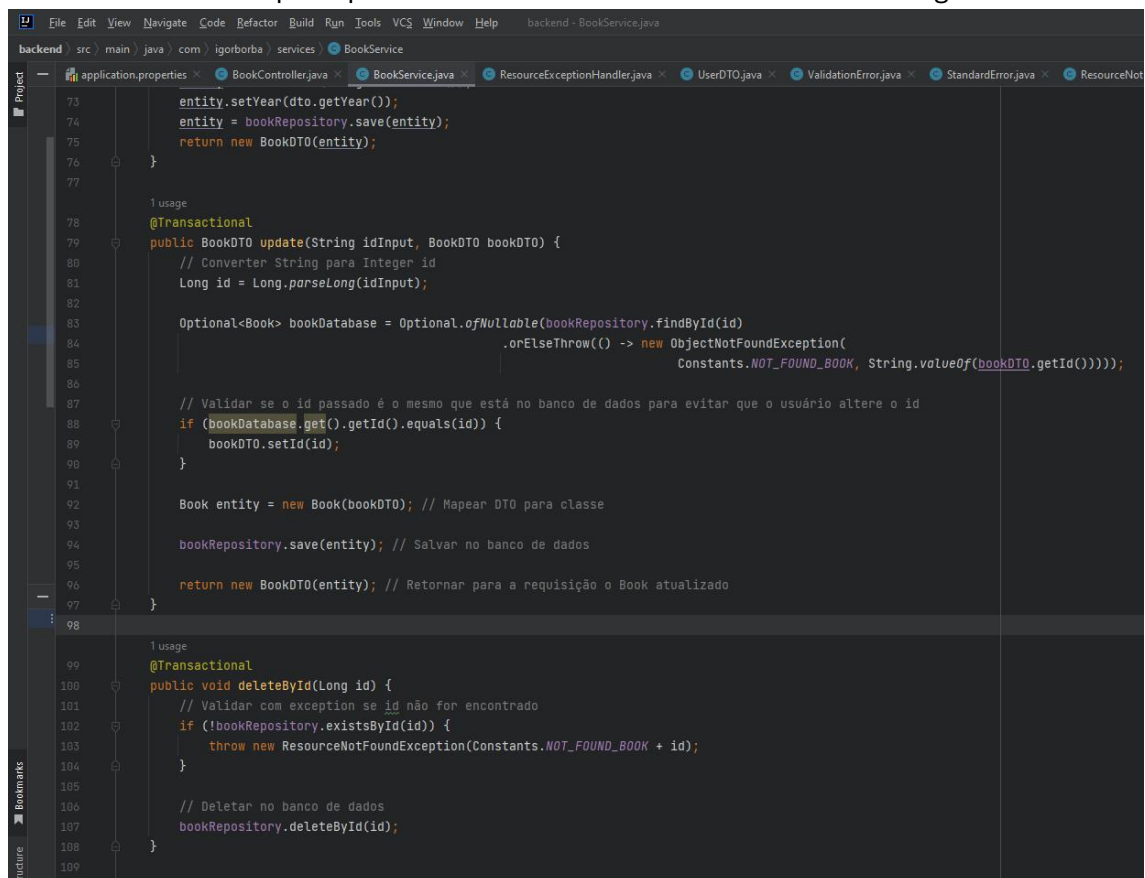


Controller com update e delete de livros.



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/BookController.java>

Service com acesso ao banco de dados, validações e Transações para possível rollback caso ocorra uma exception para não alterar o banco de dados se houver algum erro.

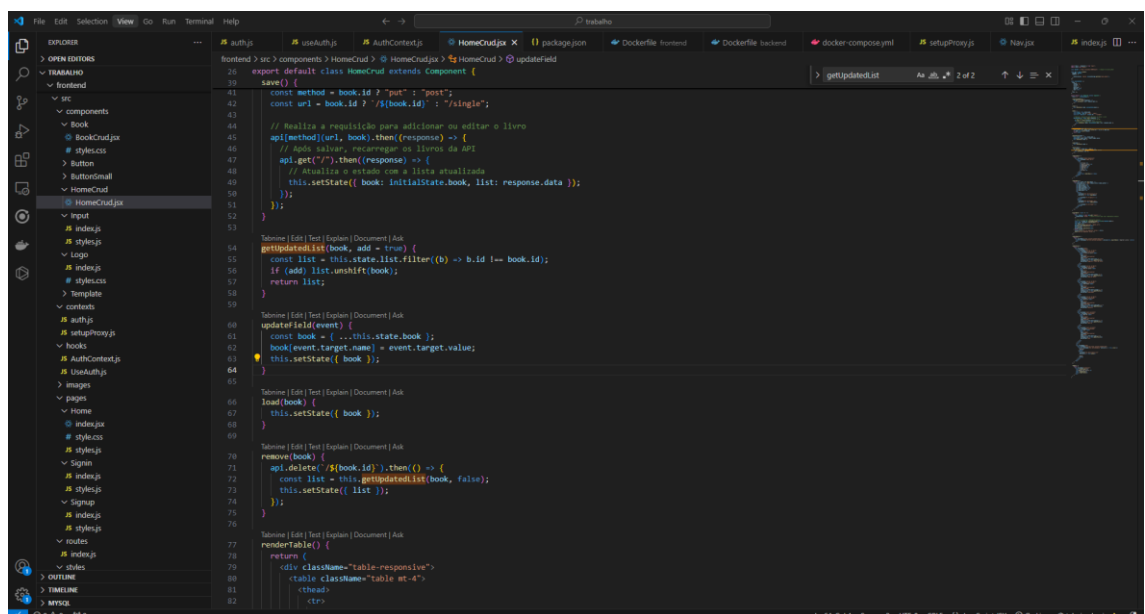


```
73     entity.setYear(dto.getYear());
74     entity = bookRepository.save(entity);
75     return new BookDTO(entity);
76 }
77
78 1 usage
79 @Transactional
80 public BookDTO update(String idInput, BookDTO bookDTO) {
81     // Converter String para Integer id
82     Long id = Long.parseLong(idInput);
83
84     Optional<Book> bookDatabase = Optional.ofNullable(bookRepository.findById(id)
85         .orElseThrow(() -> new ObjectNotFoundException(
86             Constants.NOT_FOUND_BOOK, String.valueOf(bookDTO.getId()))));
87
88     // Validar se o id passado é o mesmo que está no banco de dados para evitar que o usuário altere o id
89     if (bookDatabase.get().getId().equals(id)) {
90         bookDTO.setId(id);
91     }
92
93     Book entity = new Book(bookDTO); // Mapear DTO para classe
94     bookRepository.save(entity); // Salvar no banco de dados
95
96     return new BookDTO(entity); // Retornar para a requisição o Book atualizado
97 }
98
99 1 usage
100 @Transactional
101 public void deleteById(Long id) {
102     // Validar com exception se id não for encontrado
103     if (!bookRepository.existsById(id)) {
104         throw new ResourceNotFoundException(Constants.NOT_FOUND_BOOK + id);
105     }
106
107     // Deletar no banco de dados
108     bookRepository.deleteById(id);
109 }
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/services/BookService.java>

No update, cada campo no React é alterado o estado do objeto e o método save recebe tanto uma edição quanto uma inserção de livro (update tem id, insert não tem id).

O delete delega a responsabilidade de validação ao back-end e só altera o estado.



```
26 export default class HomeCrud extends Component {
27     save() {
28         const method = book.id ? "put" : "post";
29         const url = book.id ? `/books/${book.id}` : "/books";
30
31         // Realiza o request para adicionar ou editar o livro
32         api[method](url, book).then(response) => {
33             // Após salvar, recarregar os livros da API
34             api.get("/").then(response) => {
35                 // Atualiza o estado com a lista atualizada
36                 this.setState({ book: initialState.book, list: response.data });
37             });
38         };
39     }
40
41     // Teste | Edit | Test | Explain | Document | Ask
42     getUpdatedList(book, add = true) {
43         const list = this.state.list.filter(b => b.id !== book.id);
44         if (add) list.unshift(book);
45         return list;
46     }
47
48     // Teste | Edit | Test | Explain | Document | Ask
49     updateField(event) {
50         const book = { ...this.state.book };
51         book[event.target.name] = event.target.value;
52         this.setState({ book });
53     }
54
55     // Teste | Edit | Test | Explain | Document | Ask
56     loadBook() {
57         this.setState({ book });
58     }
59
60     // Teste | Edit | Test | Explain | Document | Ask
61     removeBook() {
62         api.delete(`/books/${book.id}`).then(() => {
63             const list = this.getUpdatedList(book, false);
64             this.setState({ list });
65         });
66     }
67
68     // Teste | Edit | Test | Explain | Document | Ask
69     renderTable() {
70         return (
71             <div className="table-responsive">
72                 <table className="table mt-4">
73                     <thead>
74                         <tr>
75                             <th>ID</th>
76                             <th>Título</th>
77                             <th>Autor</th>
78                             <th>Ano</th>
79                             <th>Ações</th>
80                         </tr>
81                     </thead>
82                     <tbody>
83                         <tr>
84                             <td>{book.id}</td>
85                             <td>{book.title}</td>
86                             <td>{book.author}</td>
87                             <td>{book.year}</td>
88                             <td>
89                                 <button className="btn btn-sm btn-info">Editar</button>
90                                 <button className="btn btn-sm btn-danger">Remover</button>
91                             </td>
92                         </tr>
93                     </tbody>
94                 </table>
95             </div>
96         );
97     }
98 }
```

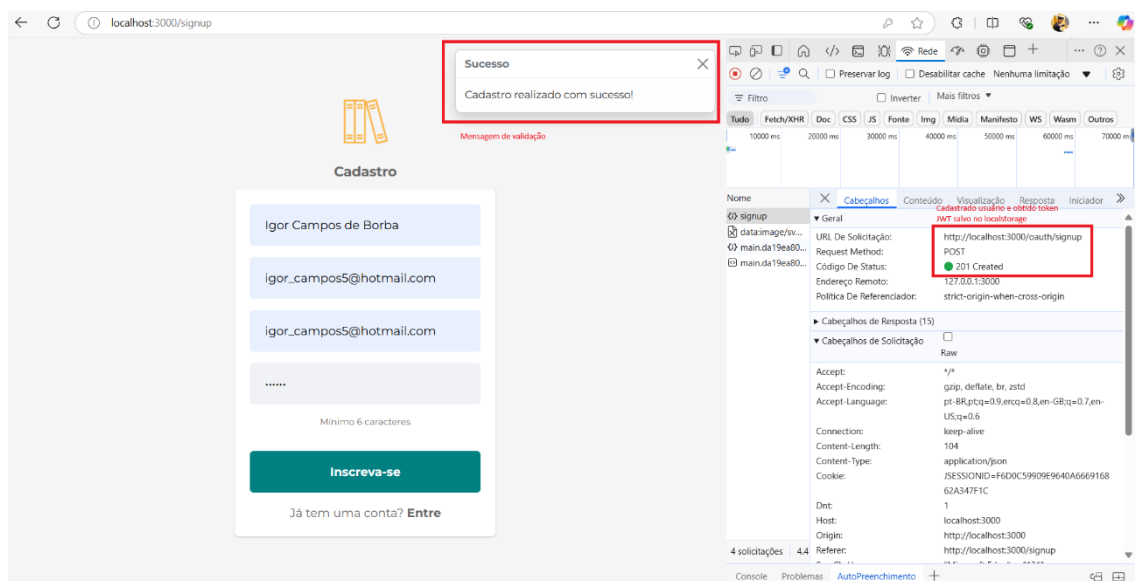
<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/components/HomeCrud/HomeCrud.jsx>

## 4 Segurança da Aplicação

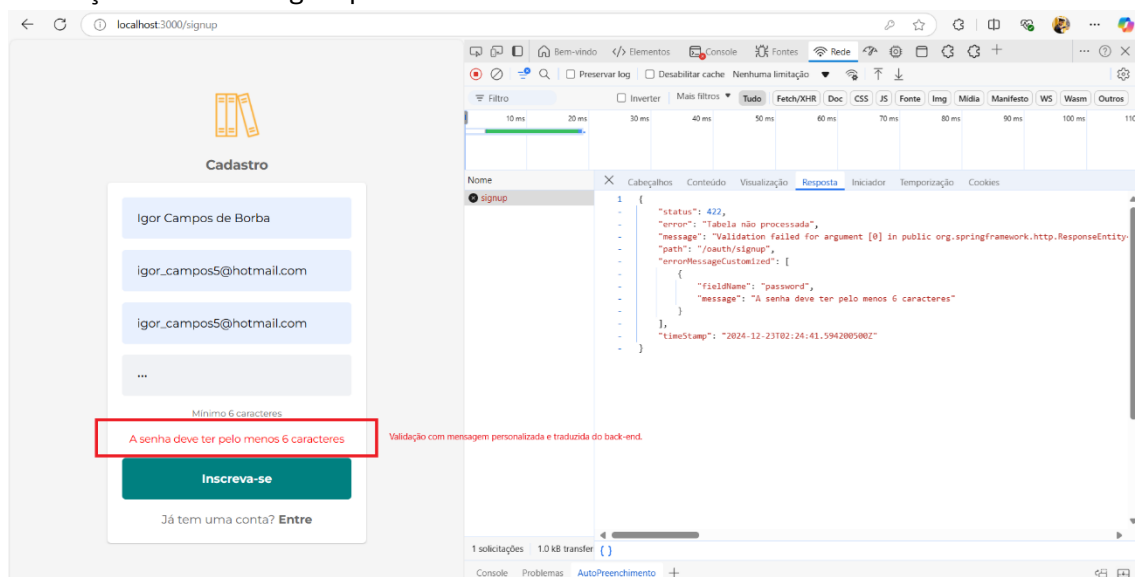
- Autenticação e Autorização com OAuth2 e Token JWT
- Validações e exceptions de integridade dos dados e de rota protegida a acessos não logados.

Utilizei OAuth2 e token JWT para o primeiro atuar como servidor de autenticação (AuthorizationServerConfig.java) e autorização de rotas segundo a role (ResourceServerConfig.java). O token JWT é enviado por meio do LoginController.java que cadastra um novo usuário e o token é configurado em SecurityConfig.java e em ResourceServerConfig.java. Inicialmente por padrão os usuários têm a role de operador e não de admin.

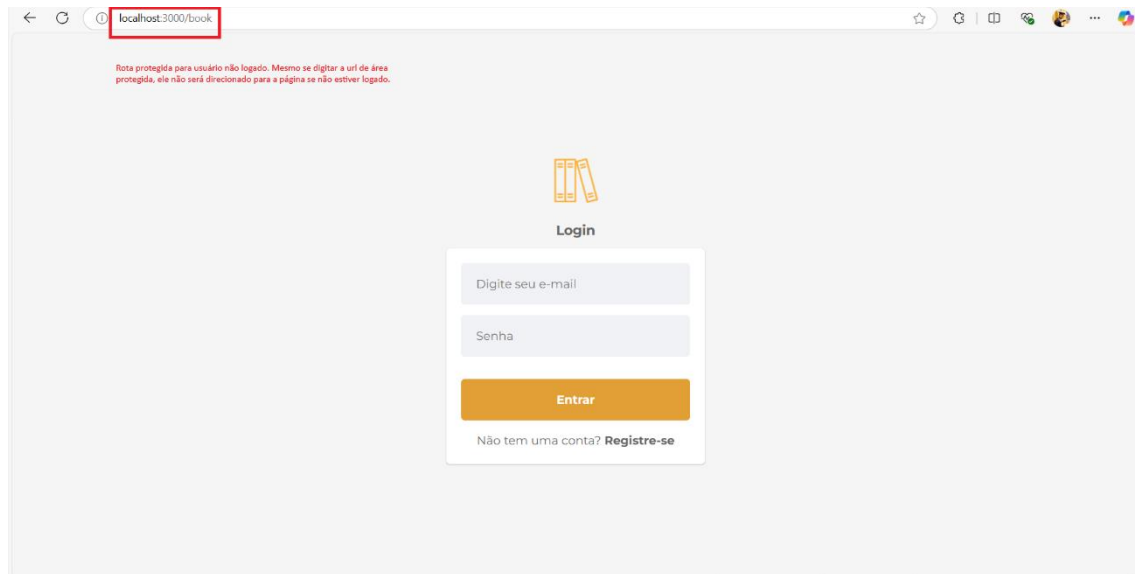
Cadastro de usuário realizado e obtido token JWT salvo no localStorage no front-end.



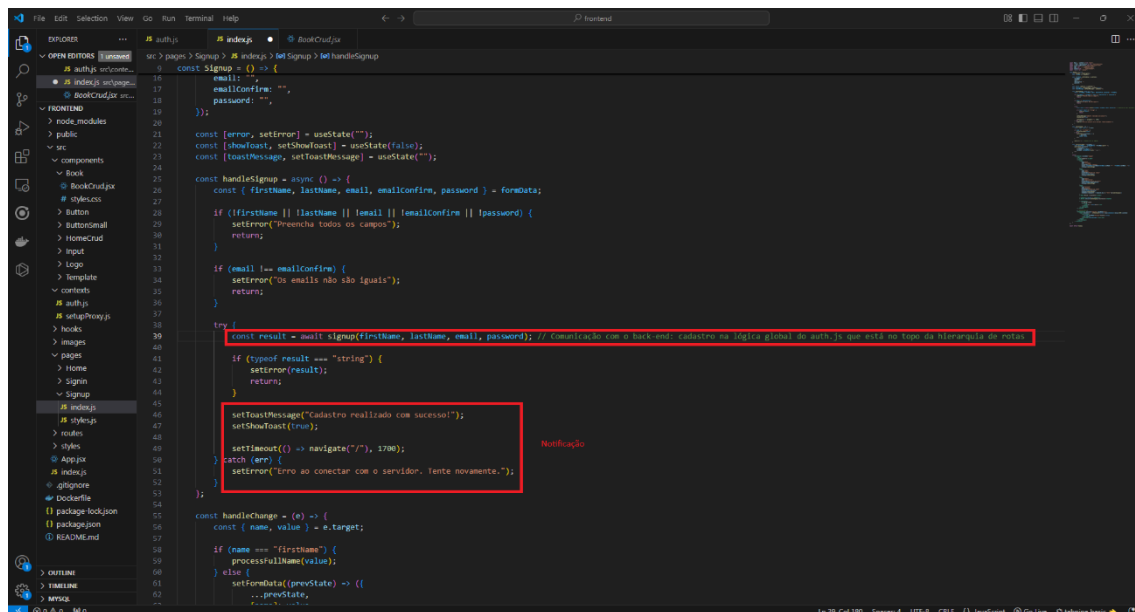
Validação com mensagem personalizada e traduzida do back-end.



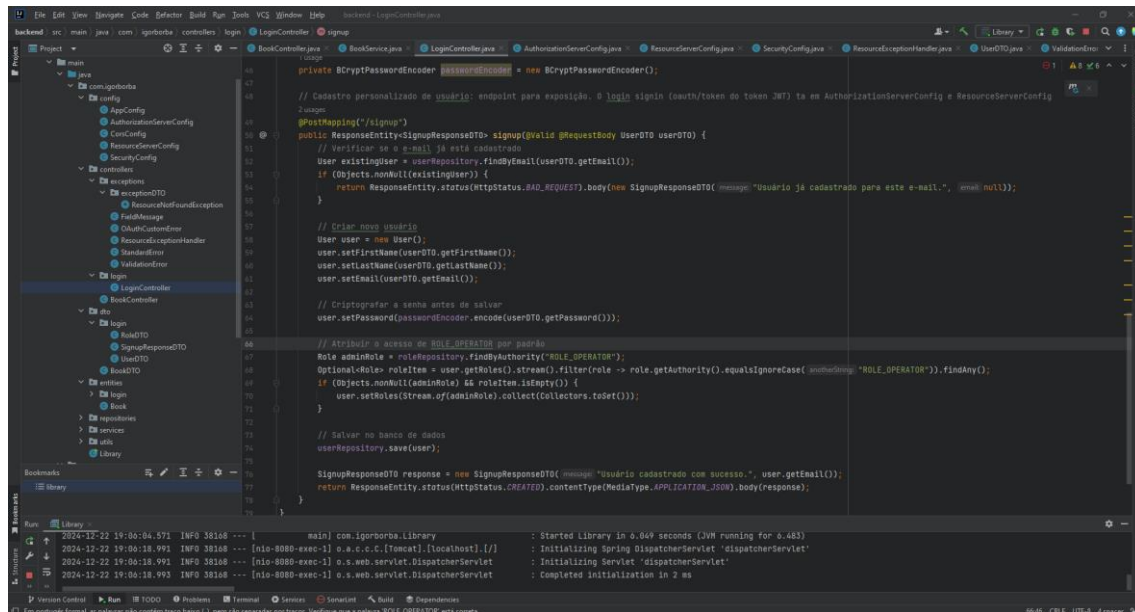
Rota protegida para usuário não logado. Mesmo se digitar a url de área protegida, ele não será direcionado para a página se não estiver logado.



A página de cadastro no React tem validações antes do envio ao back-end e depois com mensagens personalizadas do back-end.



O back-end recebe o cadastro por meio de um controller, valida no banco se usuário já está cadastrado pelo e-mail, codifica a senha com Bcrypt e atribui a role de operador a fim de ter autorização ao recursos (páginas). Também possui validação com @Valid dos atributos preenchidos no cadastro.



```
private BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

// Cadastro personalizado de usuário: endpoint para exposição. O login/signin (auth/token do token JWT) tá em AuthorizationServerConfig e ResourceServerConfig
@PostMapping("/signup")
public ResponseEntity<SignUpResponseDTO> signUp(@Valid @RequestBody UserDTO userDTO) {
    // Verificar se o e-mail já está cadastrado
    User existingUser = userRepository.findByEmail(userDTO.getEmail());
    if (Objects.nonNull(existingUser)) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new SignUpResponseDTO( "Usuário já cadastrado para este e-mail.", null));
    }

    // Criar novo usuário
    User user = new User();
    user.setFirstName(userDTO.getFirstName());
    user.setLastName(userDTO.getLastName());
    user.setEmail(userDTO.getEmail());

    // Criptografar a senha antes de salvar
    user.setPassword(passwordEncoder.encode(userDTO.getPassword()));

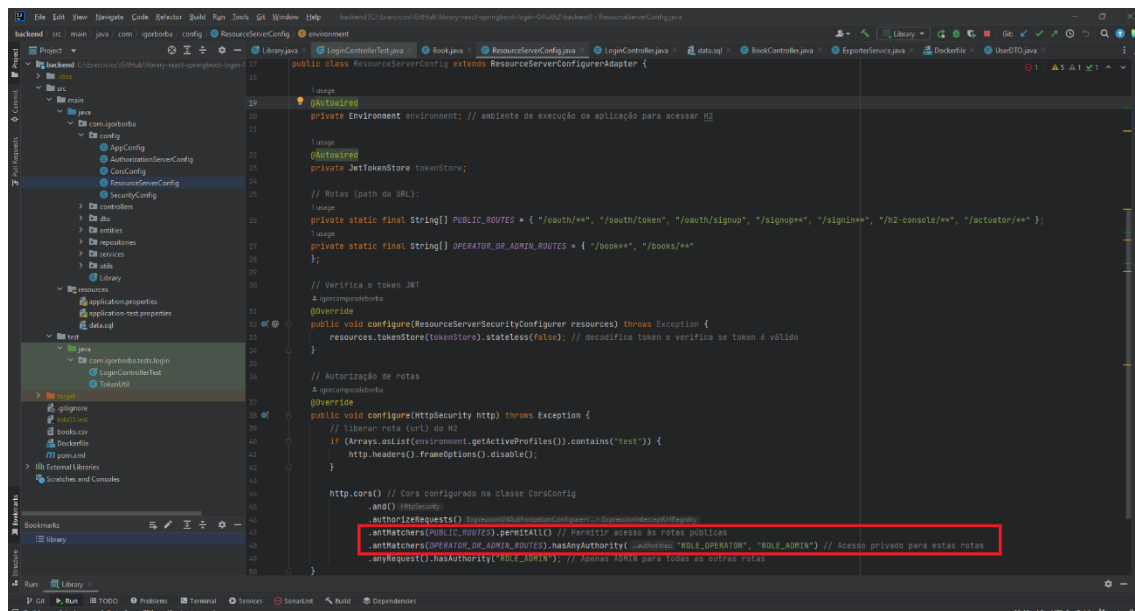
    // Atribuir o acesso de ROLE_OPERATOR por padrão
    Role adminRole = roleRepository.findByAuthority("ROLE_OPERATOR");
    Optional<Role> roleItem = user.getRoles().stream().filter(role -> role.getAuthority().equalsIgnoreCase(adminRole.getAuthority())).findAny();
    if (Objects.nonNull(adminRole) && roleItem.isEmpty()) {
        user.setRoles(Stream.of(adminRole).collect(Collectors.toList()));
    }

    // Salvar no banco de dados
    userRepository.save(user);

    SignUpResponseDTO response = new SignUpResponseDTO( "Usuário cadastrado com sucesso.", user.getEmail());
    return ResponseEntity.status(HttpStatus.CREATED).contentType(MediaType.APPLICATION_JSON).body(response);
}
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/services/login/UserService.java>

Há a configuração de acesso no servidor de autorização do OAuth2 com validação se o usuário possui a role de acesso (ROLE\_OPERATOR ou ROLE\_ADMIN). Implementadas rotas públicas e privadas.



```
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
    // Configuração de recursos
    private Environment environment; // ambiente de execução da aplicação para acessar o H2
    private OAuth2ResourceServerTokenStore tokenStore;

    // Rotas (path de URL):
    private static final String[] PUBLIC_ROUTES = { "/auth/**", "/auth/token", "/auth/signup", "/signup/**", "/signin/**", "/h2-console/**", "/actuator/**" };
    private static final String[] OPERATOR_OR_ADMIN_ROUTES = { "/book/**", "/books/**" };

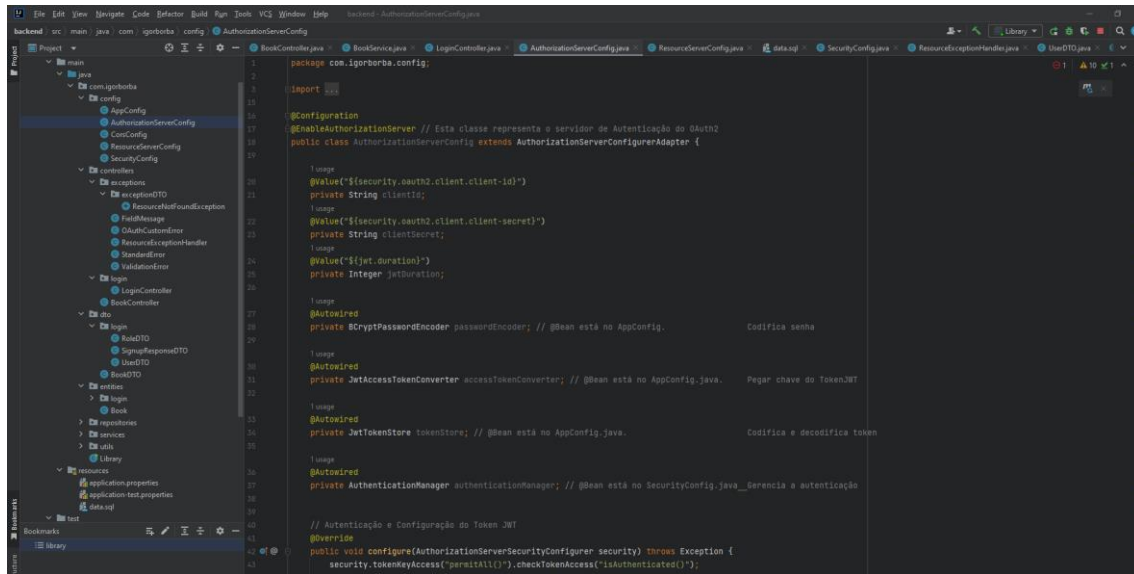
    // Verifica o token JWT
    @Override
    public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
        resources.tokenStore(tokenStore).stateless(false); // decodifica token e verifica se token é válido
    }

    // Autorização de rotas
    @Override
    public void configure(HttpSecurity http) throws Exception {
        // liberar rota (url) do H2
        if (Arrays.asList(environment.getActiveProfiles()).contains("test")) {
            http.headers().frameOptions().disable();
        }

        http.cors().and().authorizeRequests()
            .antMatchers(PUBLIC_ROUTES).permitAll() // Permite acesso as rotas publicas
            .antMatchers(OPERATOR_OR_ADMIN_ROUTES).hasAuthority("ROLE_OPERATOR", "ROLE_ADMIN") // Acesso privado para estas rotas
            .anyRequest().hasAuthority("ROLE_ADMIN"); // Apenas Admin para todas as outras rotas
    }
}
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/config/ResourceServerConfig.java>

Variáveis de ambiente protegidas dinamicamente no application.properties. E esta classe configura as credenciais OAuth2 e permissões.

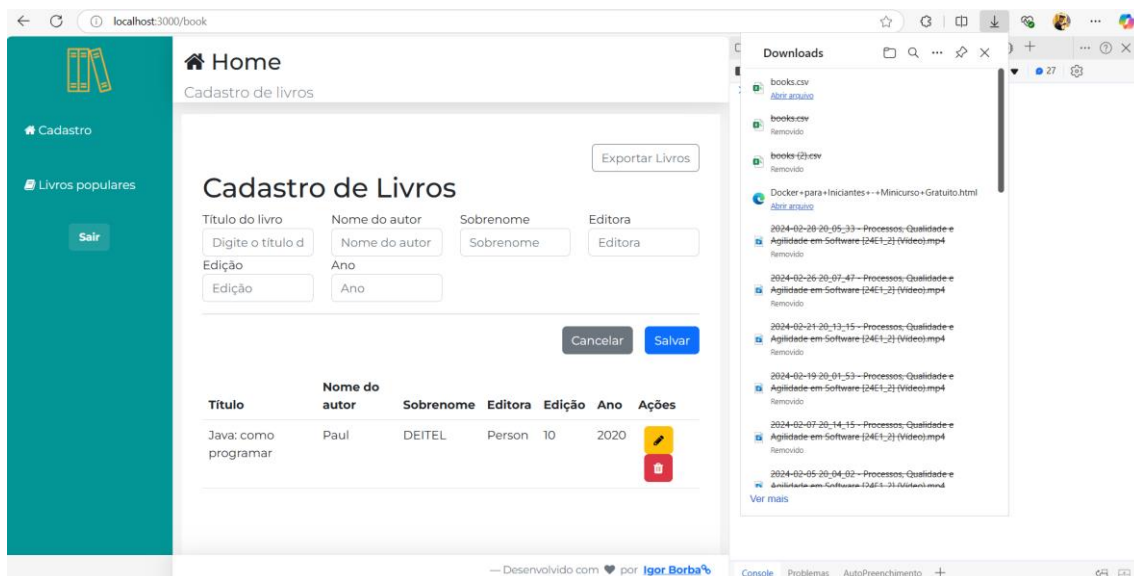


<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/config/AuthorizationServerConfig.java>

## 5 Melhorias e Segurança Avançada

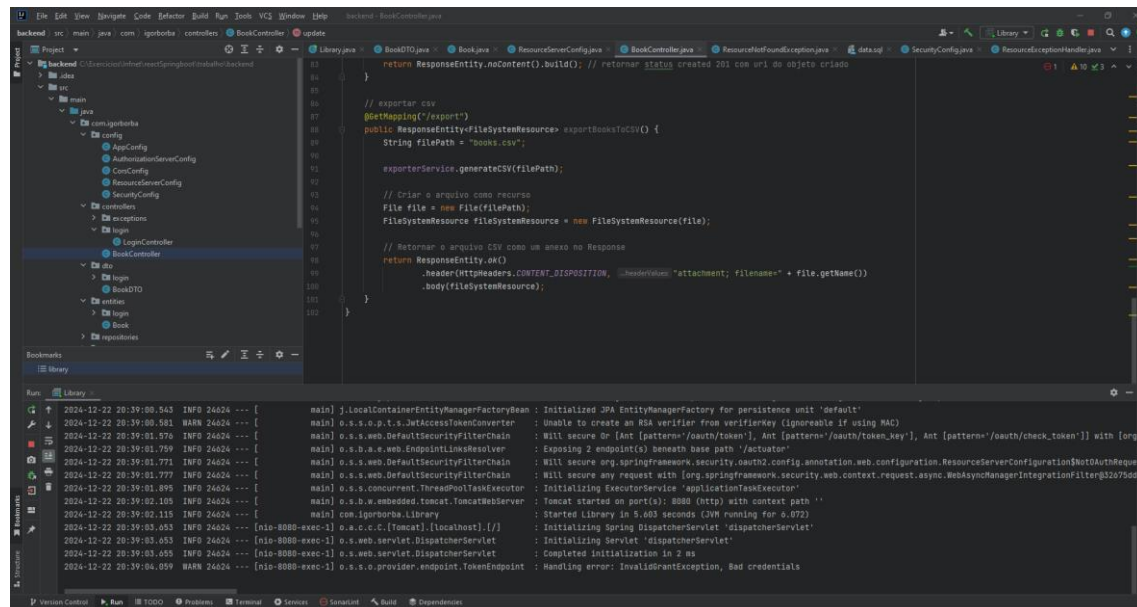
- Exportação de csv
- Senha criptografada via Bcrypt
- Logout seguro
- Clean code
- Padrão RESTful

## Exportador de csv.





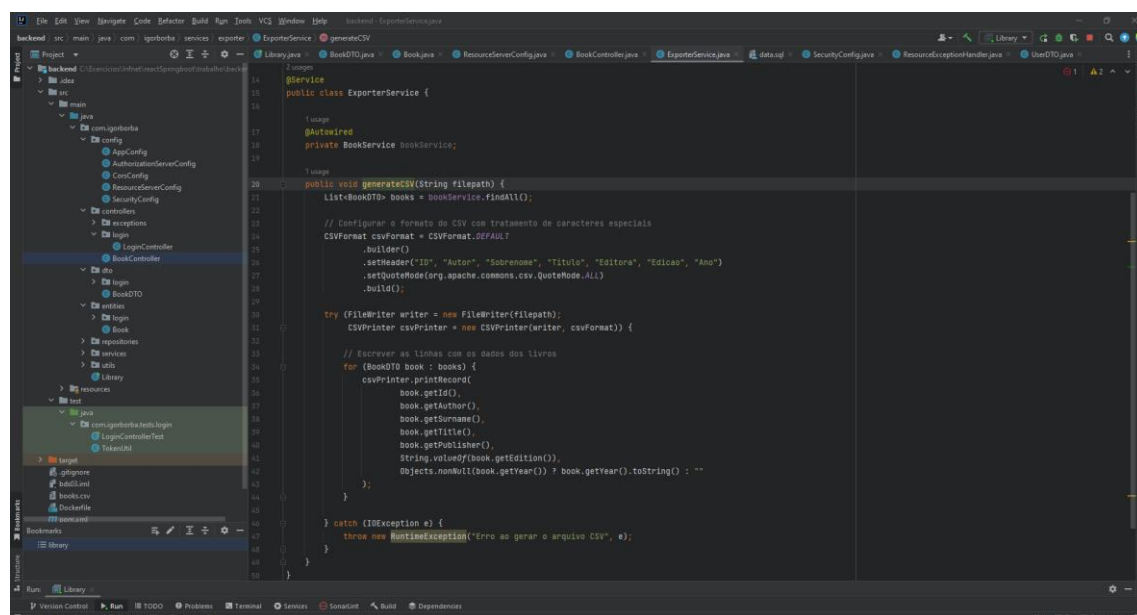
Controller do exportador de csv: chama o service que cria o arquivo e o controller encapsula para retornar como recurso ao front-end.



```
13 return ResponseEntity.noContent().build(); // retornar status created 201 com url do objeto criado
14
15 // exportar csv
16 @GetMapping("/export")
17 public ResponseEntity<FileSystemResource> exportBooksToCSV() {
18     String filePath = "books.csv";
19
20     exporterService.generateCSV(filePath);
21
22     // Criar o arquivo como recurso
23     File file = new File(filePath);
24     FileSystemResource fileSystemResource = new FileSystemResource(file);
25
26     // Retornar o arquivo CSV como um anexo no Response
27     return ResponseEntity.ok()
28         .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=" + file.getName())
29         .body(fileSystemResource);
30 }
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/BookController.java>

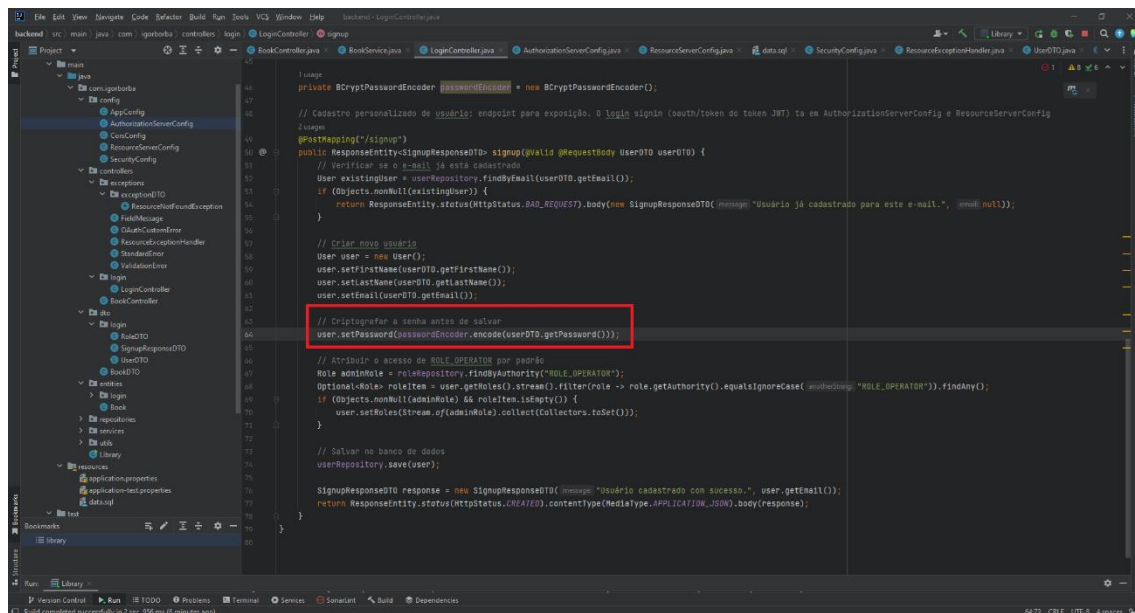
Service do exportador de csv utilizando o apache commons no service que formata o header e corpo do csv via dados obtidos no findAll de livros:



```
14 @Service
15 public class ExporterService {
16
17     @Autowired
18     private BookService bookService;
19
20     // gerar
21     public void generateCSV(String filePath) {
22         List<BookDTO> books = bookService.findAll();
23
24         // Configurar o formato do CSV com tratamento de caracteres especiais
25         CSVFormat csvFormat = CSVFormat.DEFAULT
26             .builder()
27             .setHeader("ID", "Autor", "Sobrenome", "Titulo", "Editora", "Edicao", "Ano")
28             .setQuoteMode(org.apache.commons.csv.QuoteMode.ALL)
29             .build();
30
31         try (FileWriter writer = new FileWriter(filePath);
32             CSVPrinter csvPrinter = new CSVPrinter(writer, csvFormat)) {
33
34             // Escrever as linhas com os dados dos livros
35             for (BookDTO book : books) {
36                 csvPrinter.printRecord(
37                     book.getId(),
38                     book.getAuthor(),
39                     book.getSurname(),
40                     book.getTitle(),
41                     book.getPublisher(),
42                     String.valueOf(book.getEdition()),
43                     Objects.nonNull(book.getYear()) ? book.getYear().toString() : ""
44                 );
45             }
46
47         } catch (IOException e) {
48             throw new RuntimeException("Erro ao gerar o arquivo CSV", e);
49         }
50     }
51 }
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/services/exporter/ExporterService.java>

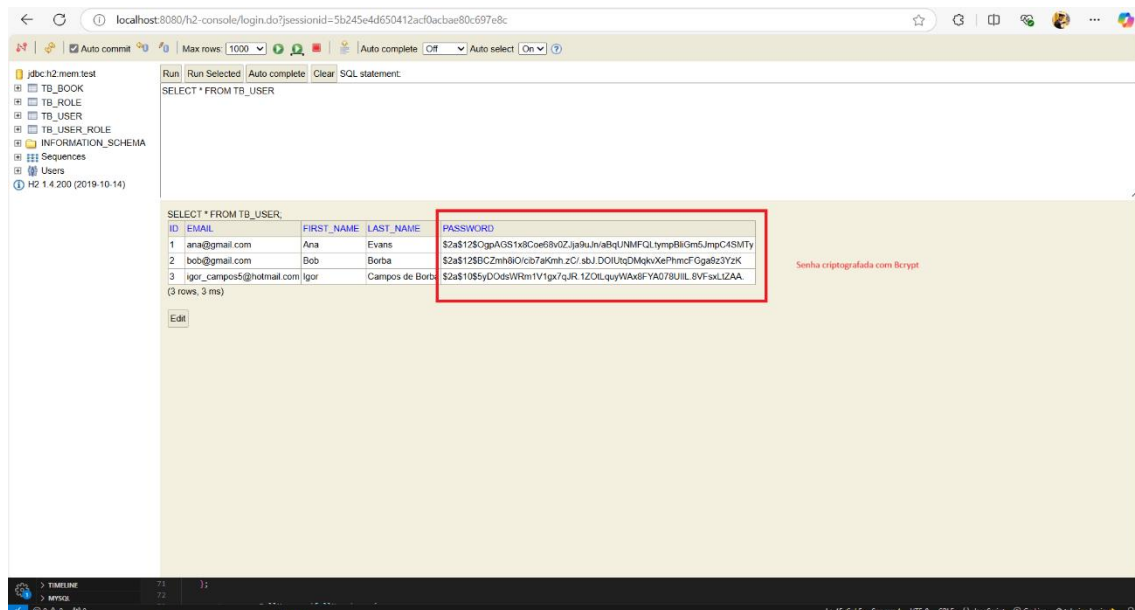
## Criptografia de senha via Bcrypt no cadastro de usuários.



```
44 private BcryptPasswordEncoder passwordEncoder = new BcryptPasswordEncoder();
45
46 // Cadastro personalizado de usuário: endpoint para exposição. O login sign up (auth/token do token JWT) tá em AuthorizationServerConfig e ResourceServerConfig
47
48 @PostMapping("/signup")
49 public ResponseEntity<SignupResponseDTO> signUp(@Valid @RequestBody UserDTO userDTO) {
50     // Verificar se o e-mail já está cadastrado
51     User existingUser = userRepository.findByEmail(userDTO.getEmail());
52     if (Objects.nonNull(existingUser)) {
53         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new SignupResponseDTO("Usuário já cadastrado para este e-mail.", null));
54     }
55
56     // Criar novo usuário
57     User user = new User();
58     user.setFirstName(userDTO.getFirstName());
59     user.setLastName(userDTO.getLastName());
60     user.setEmail(userDTO.getEmail());
61
62     // Criptografar a senha antes de salvar
63     user.setPassword(passwordEncoder.encode(userDTO.getPassword()));
64
65     // Atribuir o acesso de ROLE_OPERATOR por padrão
66     Role adminRole = userRepository.findAuthority("ROLE_OPERATOR");
67     Optional<Role> roleItem = user.getRoles().stream().filter(role -> role.getAuthority().equalsIgnoreCase(adminRole.getAuthority())).findAny();
68     if (Objects.nonNull(adminRole) && roleItem.isEmpty()) {
69         user.setRoles(Stream.of(adminRole).collect(Collectors.toList()));
70     }
71
72     // Salvar no banco de dados
73     userRepository.save(user);
74
75     SignupResponseDTO response = new SignupResponseDTO("Usuário cadastrado com sucesso.", user.getEmail());
76     return ResponseEntity.status(HttpStatus.CREATED).contentType(MediaType.APPLICATION_JSON).body(response);
77 }
78 }
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/main/java/com/igorborba/controllers/login/LoginController.java>

## Senha criptografada com Bcrypt. Entidade do banco de dados de testes h2.



ID	EMAIL	FIRST_NAME	LAST_NAME	PASSWORD
1	ana@gmail.com	Ana	Evans	\$2a\$12\$CgpgAGS1x8Coe6BvQZ.ja9uJnvaBqJUNMFGLympBkGm5JmpC4SMty
2	bob@gmail.com	Bob	Borba	\$2a\$12\$BCZmh8i0icb7aKmh.zC/sbJ.DOLUqDMqkXePhmcfGga8z3YzK
3	igor_campos5@hotmail.com	igor	Campos de Borba	\$2a\$10\$5yDodsWRm1v1px/q.R.1ZOLquyWAx8fYA078UJL6VF-sxLZAA

Senha criptografada com Bcrypt



No React, o botão de logout atribui como nulo o token e repassa a informação via contexto.

```
export const AuthProvider = ({ children }) => {
  const signin = async (firstName, lastName, email, password) => {
    const response = await fetch("http://localhost:8080/auth/signup", {
      body: JSON.stringify({ firstName, lastName, email, password }),
    });
    if (response.ok) {
      const data = await response.json();
      return data; // Retorna os dados do cadastro, como o usuário criado ou um token
    } else {
      const error = await response.json();
      return error.message || "Erro ao realizar cadastro.";
    }
  } catch (err) {
    return "Erro ao conectar ao servidor.";
  }
};

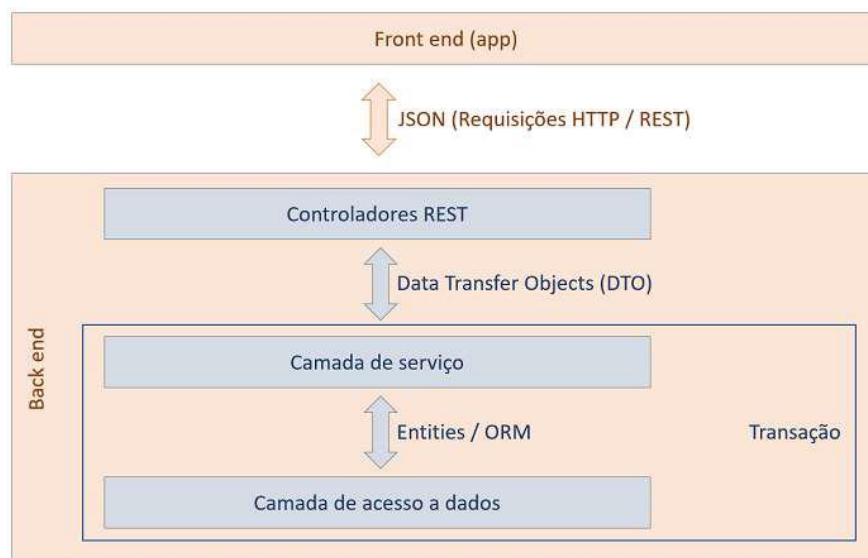
// Método para DESLOGAR usuário
const signout = () => {
  setToken(null); // Remove token
  setHeader(null);
  localStorage.removeItem("auth_token");
};

// Limpar o cabeçalho de autenticação
delete axios.defaults.headers.common["Authorization"]; // Usar delete para garantir que o header seja removido

return {
  AuthContext.Provider value={ { user, signed: !!user, signin, signup, signout } },
  {children}
} // AuthContext.Provider
};
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/contexts/auth.js>

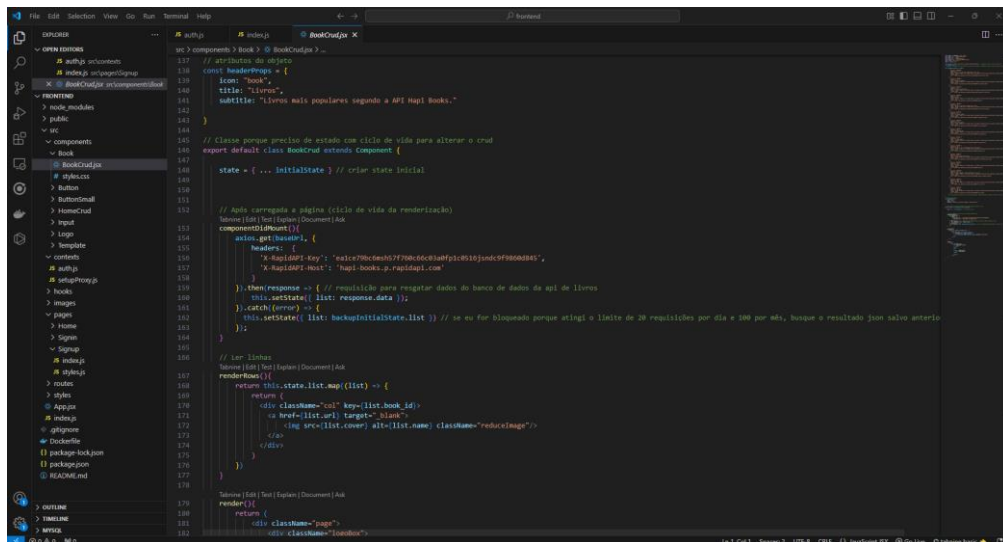
A estrutura do projeto no Spring Boot foi modularizada em camadas.



Fonte: Elaborado pelo autor.

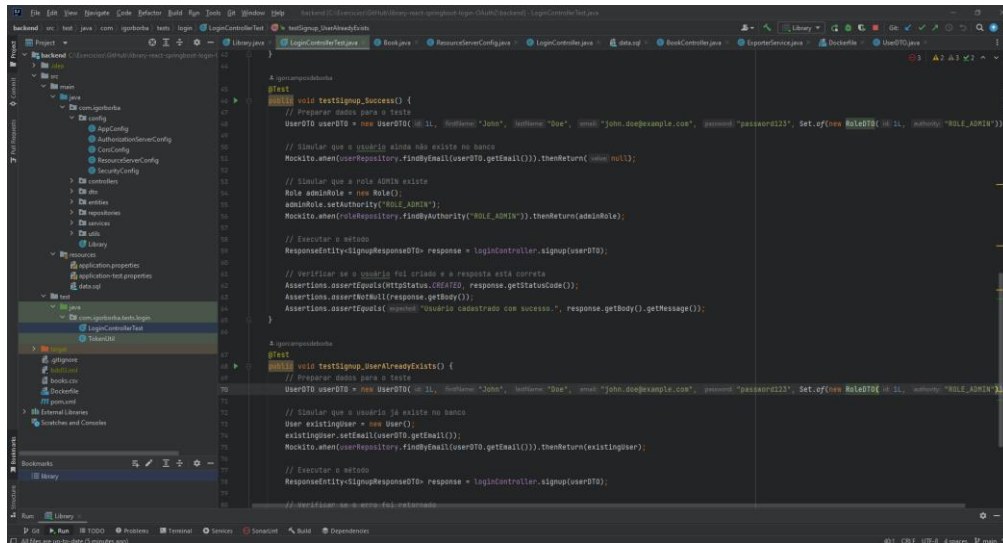
- **Front-end:** Camada de apresentação e interação desenvolvida em React, JavaScript, HTML e CSS.
- **JSON:** formato que encapsula objetos com notação JavaScript para transmitir dados entre sistemas. Exemplo: um JSON é retornado como resposta para o GET (do HTTP) do catálogo no front-end.
- **Controller REST (resource):** expõe os endpoints para receber as requisições do front-end do protocolo HTTP.
- **DTO (data transfer objects):** eu defino quais dados do banco serão transmitidos ao front-end. E separa a camada lógica do service com a apresentação do front.

- Adicional: **API de terceiros** com backup local devido ao número de 20 requisições por dia e 100 por mês.



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/src/components/Book/BookCrud.jsx>

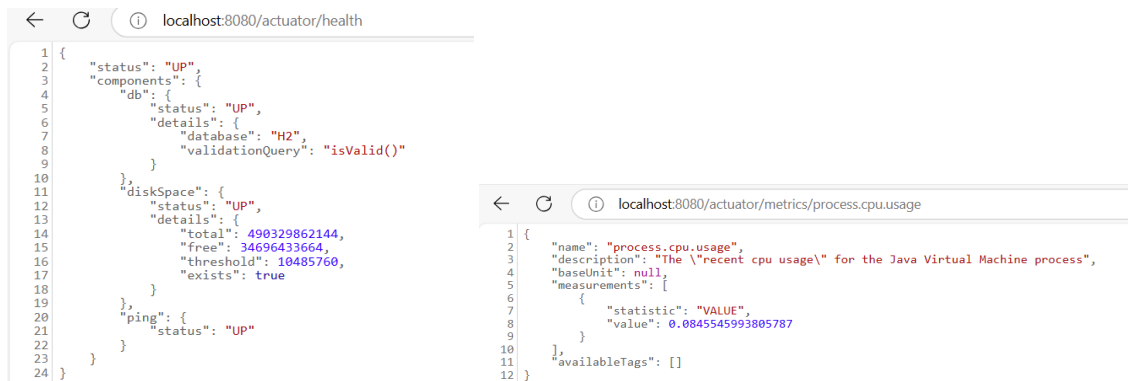
Adicional: **teste unitário** do cadastro (signin) e do token no back-end.



```
1 // ...
2
3 // ...
4
5 // ...
6
7 // ...
8
9 // ...
10
11 // ...
12
13 // ...
14
15 // ...
16
17 // ...
18
19 // ...
20
21 // ...
22
23 // ...
24
25 // ...
26
27 // ...
28
29 // ...
30
31 // ...
32
33 // ...
34
35 // ...
36
37 // ...
38
39 // ...
40
41 // ...
42
43 // ...
44
45 // ...
46
47 // ...
48
49 // ...
50
51 // ...
52
53 // ...
54
55 // ...
56
57 // ...
58
59 // ...
60
61 // ...
62
63 // ...
64
65 // ...
66
67 // ...
68
69 // ...
70
71 // ...
72
73 // ...
74
75 // ...
76
77 // ...
78
79 // ...
80
81 // ...
82
83 // ...
84
85 // ...
86
87 // ...
88
89 // ...
90
91 // ...
92
93 // ...
94
95 // ...
96
97 // ...
98
99 // ...
100
```

<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/src/test/java/com/igorborba/tests/login/LoginControllerTest.java>

Adicional: **atuator** para saber se a aplicação está no ar e métricas da máquina.  
Útil para configurar relatórios e alertas.



```
1 {
2   "status": "UP",
3   "components": {
4     "db": {
5       "status": "UP",
6       "details": {
7         "database": "H2",
8         "validationQuery": "isValid()"
9       }
10    }
11  },
12  "diskSpace": {
13    "status": "UP",
14    "details": {
15      "total": 490329862144,
16      "free": 34696433664,
17      "threshold": 10485760,
18      "exists": true
19    }
20  },
21  "ping": {
22    "status": "UP"
23  }
24 }
```

```
1 {
2   "name": "process.cpu.usage",
3   "description": "The \"recent cpu usage\" for the Java Virtual Machine process",
4   "baseUnit": null,
5   "measurements": [
6     {
7       "statistic": "VALUE",
8       "value": 0.084554593805787
9     }
10  ],
11   "availableTags": []
12 }
```

## 6

**Etapa opcional: Implantação, Containers e Deploy do Frontend**

- Docker compose implementado.
- Não implementado: deploy no Netlify e na AWS

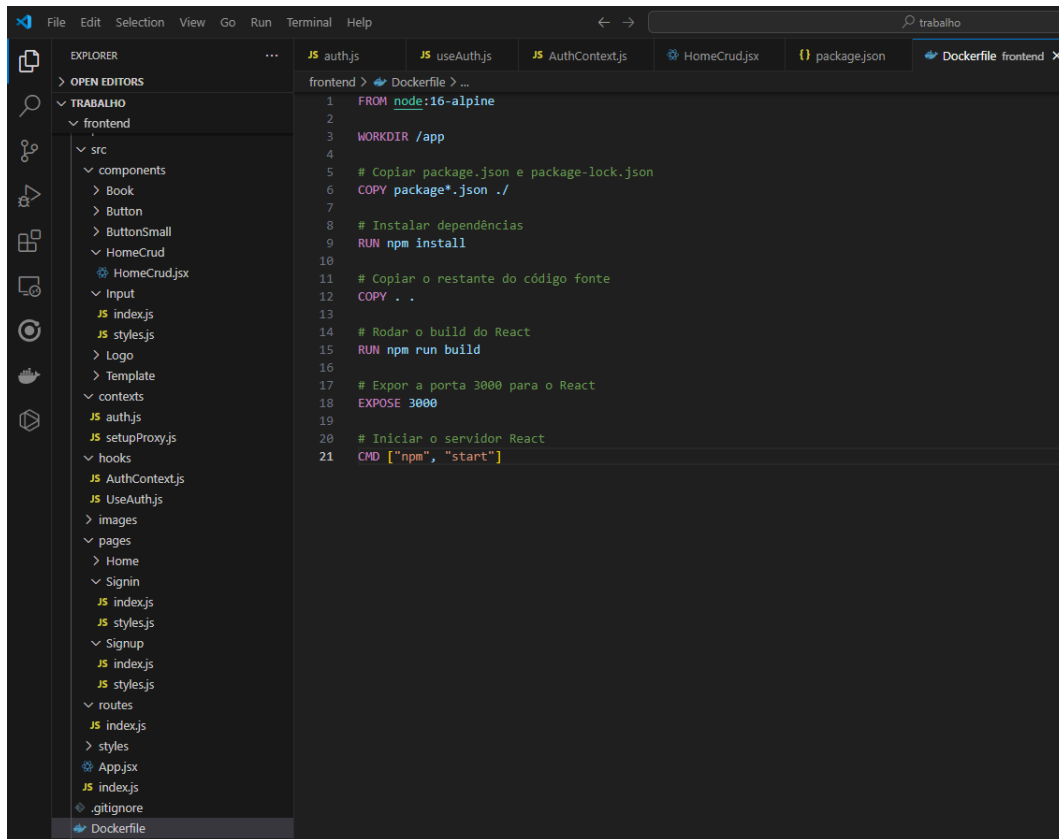
O docker compose, docker file do React e Spring foram implementados. Mas não o spring deu um erro no manifest.mf que não consegui corrigir.

Docker-compose:

```
1 version: "1.0"
2
3 services:
4   backend:
5     build:
6       context: ./backend
7     ports:
8       - "8080:8080"
9     environment:
10      - SPRING_PROFILES_ACTIVE=test
11     networks:
12      - app-network
13
14   frontend:
15     build:
16       context: ./frontend
17     ports:
18       - "3000:3000"
19     networks:
20       - app-network
21
22 networks:
23   app-network:
24     driver: bridge
```

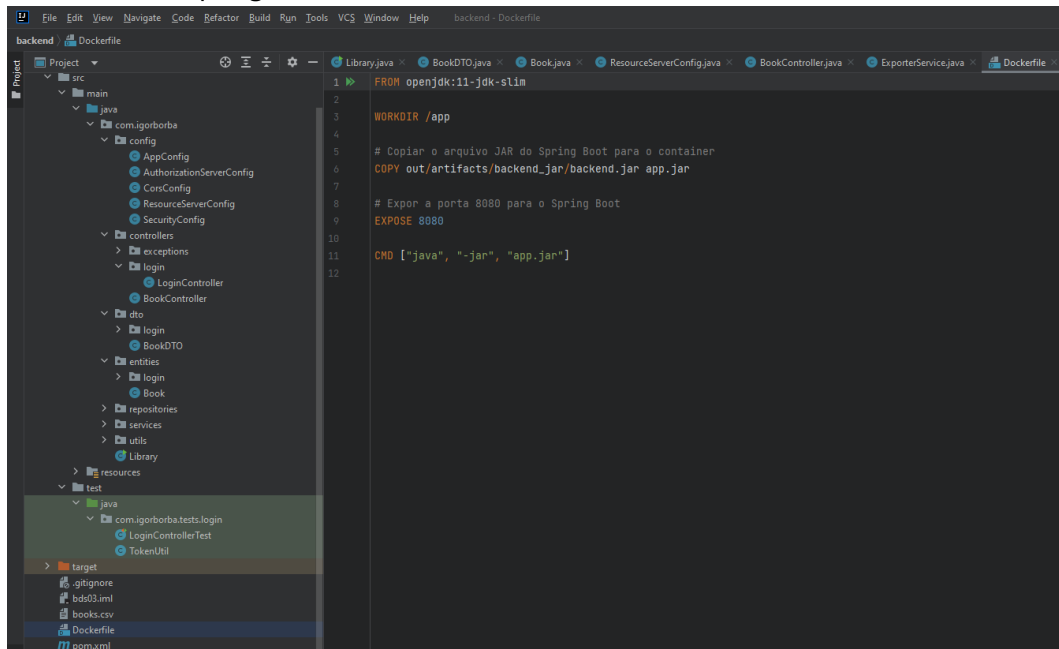
<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/docker-compose.yml>

Dockerfile do React:



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/frontend/Dockerfile>

### Dockerfile do Spring Boot:



<https://github.com/igorcamposdeborba/library-react-springboot-login-OAuth2/blob/main/backend/Dockerfile>