

Projeto BlogBosch

Docupedia Export

Author:Ferro Alisson (CtP/ETS)

Date:13-Sep-2023 13:19

Table of Contents

1 Descrição	4
2 Front-end	5
2.1 Criando o projeto em React	5
2.2 Configurando as rotas	5
2.3 Criando componentes	7
2.4 Criando pagina de adicionar	9
2.5 Criando página de login	11
2.6 Criando página para se registrar	16
2.7 Criando AccessDenied	19
2.8 Criando Protected Route	20
3 Back-end	23
3.1 Criando o projeto	23
3.2 Configurando o banco de dados	24
3.3 Configurando as Rotas	26
3.4 Criando o controller	28
3.5 Criando operação de curtir um artigo	31
3.6 Registrando Usuário	32
3.7 Login do usuário	36
4 Integrando o front-end com API	37
4.1 Exibindo os artigos	37
4.2 Curtindo um artigo	39
4.3 Criando um autor	39
4.4 Login do usuário	41
4.5 Protegendo as Rotas	43

4.6 Criando AccessDenied	44
4.7 Criando um artigo	45
4.8 Paginação	47
4.9 Agora é com você	52

1 Descrição

Nome do Projeto: BlogBosch - Plataforma de Blog

Descrição: O BlogBosch é uma plataforma de blog onde os usuários podem criar e publicar seus próprios artigos. O sistema permite que os usuários se registrem, façam login, escrevam artigos, visualizem e comentem em artigos de outros usuários.

Requisitos Funcionais:

1. Autenticação de Usuário:

- Os usuários devem poder se registrar com um nome de usuário, endereço de e-mail e senha.
- Os usuários devem poder fazer login usando suas credenciais registradas.

A autenticação deve ser implementada usando tokens JWT (JSON Web Tokens) para proteger as rotas e verificar a identidade do usuário.

1. Criação de Artigos:

- Os usuários autenticados devem poder criar, editar e excluir seus próprios artigos.
- Cada artigo deve ter um título, conteúdo, categoria e tags associadas.
- Os artigos devem ser armazenados no MongoDB, com referências ao autor (usuário) correspondente.

2. Visualização de Artigos:

- Os usuários devem poder visualizar todos os artigos disponíveis, listados em ordem cronológica inversa (do mais recente ao mais antigo).
- Os usuários devem poder filtrar os artigos por categoria ou tags.
- Cada artigo deve mostrar o título, conteúdo, autor, data de publicação e comentários associados.

Requisitos Técnicos:

- O back-end do sistema deve ser desenvolvido usando Node.js e Express.js.
- O banco de dados MongoDB deve ser utilizado para armazenar os usuários e artigos.
- O front-end do sistema deve ser desenvolvido usando React para a criação das interfaces de usuário.
- A comunicação entre o front-end e o back-end deve ser feita através de uma API RESTful.

2 Front-end

2.1 Criando o projeto em React

Como vimos nas aulas anteriores, para criar um projeto em react vamos utilizar o código

```
npx create-react-app blog
```

Após criar vamos utilizar algumas bibliotecas, para isso instalamos com

```
npm install react-router-dom localforage match-sorter sort-by react-bootstrap bootstrap sass
```

após isso adicionaremos a importação do bootstrap no **index.js dentro da pasta blog**

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

2.2 Configurando as rotas

Agora vamos configurar os arquivos de rotas, para isso, na pasta raiz do projeto em **index.js**, vamos colocar o componente BrowserRouter envolvendo o App.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import reportWebVitals from './reportWebVitals';
import 'bootstrap/dist/css/bootstrap.min.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
```

```
    <App />
  </BrowserRouter>
</React.StrictMode>
);

reportWebVitals();
```

Após isso, ainda na pasta raiz, em **App.js**

```
import './App.css';
import NavBar from './components/navbar';
import { Route, Routes } from 'react-router-dom';
import HomePage from './pages/home';

function App() {
  return (
    <>
      <Routes>
        <Route path='/home' element={<HomePage />} />
      </Routes>
    </>
  );
}

export default App;
```

Após isso, vamos criar a primeira página, a nossa página home
Vamos criar um arquivo em **src/pages/home/index.js**

```
import Post from "../../components/Post";

export default function HomePage(){
  return (
    <>
      <Post />
    </>
  );
}
```

```
)  
}
```

2.3 Criando componentes

Agora precisamos criar nosso arquivo **Post**, em **src/components/Post/index.js**

```
import { useEffect, useState } from 'react';  
import {  
  Button,  
  Card,  
  Container  
} from 'react-bootstrap'  
import { AiOutlineLike } from 'react-icons/ai'  
import styles from './style.module.scss';  
  
export default function Post(){  
  var [artigos, setArtigos] = useState([]);  
  
  function getPosts(){  
    setArtigos([  
      {  
        id: 1,  
        title: 'teste 1',  
        text: 'Teste',  
        likes: 10  
      },  
      {  
        id: 2,  
        title: 'teste 2',  
        text: 'Teste 2',  
        likes: 5  
      },  
    ])  
  }  
  
  useEffect(() => {  
    getPosts();  
  });  
}
```

```
}, [])

const RenderPosts = () => {
  return artigos.map((artigo) => {
    return(
      <Card key={artigo.id} className={styles.card} >
        <Card.Title className={styles.card__title}>
          {artigo.title}
        </Card.Title>
        <Card.Body className={styles.card__body}>
          <Card.Text className={styles.card__body__article}>{artigo.text}</Card.Text>
          <div className='d-flex align-items-center '>
            {artigo.likes}<Button variant='light'><AiOutlineLike /></Button>
          </div>
        </Card.Body>
      </Card>
    )
  })
}

return(
  <Container>
    <RenderPosts />
  </Container>
)
}
```

Crie agora um arquivo de estilos e coloque em **src/components/Post/styles.module.scss** e dê a estilização que achar melhor. Agora vamos criar a Navbar para o projeto em **src/componentes/NavBar/index.js**

```
import Container from "react-bootstrap/Container";
import Nav from "react-bootstrap/Nav";
import Navbar from "react-bootstrap/Navbar";
import { Link } from "react-router-dom";
import styles from './styles.module.scss';

export default function NavBar() {
```



```
return (  
  <Navbar expand="lg">  
    <Container fluid>  
      <Navbar.Toggle aria-controls="navbarScroll" />  
      <Navbar.Collapse id="navbarScroll">  
        <Nav  
          className="me-auto my-2 my-lg-0"  
          style={{ maxHeight: "100px" }}  
          navbarScroll  
        >  
          <div className={styles.links}>  
            <Link to="/home" className={styles.links__link}>Home</Link>  
            <Link to="/add" className={styles.links__link}>Adicionar</Link>  
          </div>  
        </Nav>  
      </Navbar.Collapse>  
    </Container>  
  </Navbar>  
}
```

E da mesma forma, dê a estilização da melhor forma que achar necessário

2.4 Criando pagina de adicionar

Crie um arquivo em **src/pages/AddPost/index.js**

```
import Formulario from "../../components/Formulario";  
import NavBar from "../../components/navbar";  
  
export default function AddPostPage(){  
  return (  
    <>  
      <NavBar />  
      <Formulario />  
    </>  
  )  
}
```

Agora vamos criar o arquivo **Formulário** em **src/components/Formulario/index.js**

```
import { useState } from "react";
import {
  Button,
  Col,
  Container,
  Form,
  Row
} from "react-bootstrap";
import styles from './styles.module.scss';

export default function Formulario(){
  var [author, setAuthor] = useState('');
  var [title, setTitle] = useState('');
  var [text, setText] = useState('');

  return(
    <Container>
      <Row>
        <Col>
          <Form onSubmit={handleSubmit} className={styles.form}>
            <Form.Text className={styles.form__title}>Digite Aqui seu Artigo</Form.Text>
            <Form.Control
              placeholder="Autor"
              value={author}
              onChange={(e) => setAuthor(e.target.value)}
            />
            <Form.Control
              placeholder="Titulo"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
            />
            <Form.Control
              as='textarea'
              placeholder="Texto"
              rows={5}
            />
          </Form>
        </Col>
      </Row>
    </Container>
  );
}
```

```
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
      <Col xs={12} sm={9} md={6} className={styles.form__div}>
        <Button type="submit" className={styles.form__div__button} >Salvar</Button>
      </Col>
    </Form>
  </Col>
</Row>
</Container>
)
}
```

E dê a estilização que achar mais conveniente

2.5 Criando página de login

Para criarmos a página de Login, vamos criar um arquivo em **src/components/CardLogin/index.js**. por enquanto o submit só vai navegar para outra página, posteriormente, vamos integrar com a API será necessário alterar o método

```
import { useContext, useState } from "react";
import { Button, Card, Form } from "react-bootstrap";
import { useNavigate } from 'react-router-dom';
import styles from './styles.module.scss';
import { AlertContext } from "../../context/alert";

export default function CardLogin(){
  const { setMessage, setShow, setVariant } = useContext(AlertContext);

  const navigate = useNavigate();
  var [email, setEmail] = useState('');
  var [pass, setPass] = useState('');

  function handleSubmit(e){
    e.preventDefault();
    if(!formValid()) return

    navigate('/home')
```

```
}

function formValid(){
  if(!email.includes('@')){
    setMessage('Insira um e-mail válidos')
    setShow(true);
    setVariant('danger')
    return false;
  }
  if(email.length < 5){
    setMessage('Insira um e-mail válido')
    setShow(true);
    setVariant('danger')
    return false;
  }

  return true
}

return(
  <Card className={styles.card}>
    <Card.Header className={styles.card__header}>
      <Card.Title>Login</Card.Title>
    </Card.Header>
    <Card.Body>
      <Form
        className={styles.card__form}
        onSubmit={handleSubmit}
      >
        <Form.Control
          value={email}
          placeholder="Insira seu e-mail"
          onChange={(e) => setEmail(e.target.value)}
        />
        <Form.Control
          value={pass}
          placeholder="Insira sua senha"
          onChange={(e) => setPass(e.target.value)}
        />
      </Form>
    </Card.Body>
  </Card>
)
```

```
        <Button
          className={styles.card__form__button}
          type='submit'
        >
          Entrar
        </Button>
      </Form>
    </Card.Body>
  </Card>
)
}
```

E criar um arquivo em **src/pages/Login/index.js**

```
import { Col, Row } from 'react-bootstrap';
import CardLogin from '../../components/CardLogin';
import styles from './styles.module.scss';
import AlertComponent from '../../components/Alert';

export default function LoginPage(){
  return(
    <Col className={styles.container}>
      <Row className={styles.container__row}>
        <Col xs={12} sm={8} md={4}>
          <AlertComponent />
          <CardLogin />
        </Col>
      </Row>
    </Col>
  )
}
```

Agora para criar o component **AlertComponent** em **src/components/AlertComponent/index.js**

```
import { useContext } from "react";
import { Alert } from "react-bootstrap";
```

```
import { AlertContext } from "../../context/alert";
import styles from './styles.module.scss';

export default function AlertComponent(){
  const { message, variant, show, setShow } = useContext(AlertContext);

  return(
    <Alert
      show={show}
      variant={variant}
      onClose={() => setShow(false)}
      dismissible
    >
      <p className={styles.alert}>{message}</p>
    </Alert>
  )
}
```

Falta criarmos o **AlertContext**, em **src/context/alert/index.js**

```
import React, { useEffect, useState } from "react";

export const AlertContext = React.createContext();
AlertContext.displayName = 'Alert';

export const AlertProvider = ({ children }) => {
  var [message, setMessage] = useState('');
  var [variant, setVariant] = useState('danger');
  var [show, setShow] = useState(false);

  async function handleShow(){
    setTimeout(() => {
      setShow(false)
    }, 5000)
  }

  useEffect(() => {
    handleShow()
  })
}
```

```
    }, [show])

    return(
      <AlertContext.Provider
        value={{
          message,
          setMessage,
          variant,
          setVariant,
          show,
          setShow
        }}
      >
        {children}
      </AlertContext.Provider>
    )
  }
}
```

Por fim, para funcionar o context precisamos importa-lo nas rotas, então em **App.js**

```
import { Route, Routes } from 'react-router-dom';
import HomePage from './pages/home';
import AddPostPage from './pages/AddPost';
import LoginPage from './pages/Login';
import RegisterPage from './pages/Register';
import { AlertProvider } from './context/alert';
import './App.css';

function App() {
  return (
    <>
      <AlertProvider>
        <Routes>
          <Route path="/" element={<LoginPage />} />
          <Route path="/home" element={<HomePage />} />
          <Route path="/add" element={<AddPostPage />} />
          <Route path="/register" element={<RegisterPage />} />
        </Routes>
      </AlertProvider>
    </>
  );
}
```

```
    </AlertProvider>
  </>
);
}
```

```
export default App;
```

2.6 Criando página para se registrar

Vamos criar um arquivo em **src/pages/Register/index.js**

```
import { Col, Row } from "react-bootstrap";
import styles from './styles.module.scss';
import CardRegister from "../../components/CardRegister";
import AlertComponent from "../../components/AlertComponent";

export default function RegisterPage(){
  return(
    <Col className={styles.container}>
      <Row className={styles.container__row}>
        <Col xs={12} sm={8} md={4}>
          <AlertComponent />
          <CardRegister />
        </Col>
      </Row>
    </Col>
  )
}
```

Em **src/components/CardRegister/index.js** vamos criar o **CardRegister**

```
import { useContext, useState } from "react";
import {
  Button,
  Card,
  Form
} from "react-bootstrap";
```



```
import axios from 'axios';
import styles from './styles.module.scss';
import { AlertContext } from "../../context/alert";

export default function CardRegister(){
  const { setMessage, setShow, setVariant } = useContext(AlertContext);

  var [name, setName] = useState('');
  var [email, setEmail] = useState('');
  var [birth, setBirth] = useState(Date());
  var [password, setPassword] = useState('');
  var [confirmPass, setConfirmPass] = useState('');

  function handleSubmit(e){
    e.preventDefault();
    if(!formValid()) return
  }

  function formValid(){
    if(!name.includes(' ')){
      setMessage('Insira nome e sobrenome')
      setShow(true);
      setVariant('danger')
      return false;
    }
    if(name.length<5){
      setMessage('Insira um nome e sobrenome válidos')
      setShow(true);
      setVariant('danger')
      return false;
    }
    if(!email.includes('@')){
      setMessage('Insira um e-mail válidos')
      setShow(true);
      setVariant('danger')
      return false;
    }
    if(email.length < 5){
      setMessage('Insira um e-mail válido')
```

```
        setShow(true);
        setVariant('danger')
        return false;
    }
    if(confirmPass !== password) {
        setMessage('As senhas não conferem')
        setShow(true);
        setVariant('danger')
        return false;
    }
    if(password.length < 6) {
        setMessage('Senha inferior a 6 caracteres')
        setShow(true);
        setVariant('danger')
        return false
    };

    return true
}

return(
    <Card className={styles.card}>
        <Card.Header className={styles.card__header}>
            <Card.Title>Registrar-se</Card.Title>
        </Card.Header>
        <Card.Body>
            <Form
                className={styles.card__form}
                onSubmit={handleSubmit}
            >
                <Form.Label>Insira seu nome</Form.Label>
                <Form.Control
                    placeholder="Nome Completo"
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                />
                <Form.Label>Insira seu e-mail</Form.Label>
                <Form.Control
                    placeholder="E-mail"
```

```
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <Form.Label>Insira sua data de nascimento</Form.Label>
      <Form.Control
        type="date"
        value={birth}
        onChange={(e) => setBirth(e.target.value)}
      />
      <Form.Label>Insira sua senha</Form.Label>
      <Form.Control
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <Form.Label>Confirme sua senha</Form.Label>
      <Form.Control
        type="password"
        value={confirmPass}
        onChange={(e) => setConfirmPass(e.target.value)}
      />
      <Button
        className={styles.card__form__button}
        type='submit'
      >
        Entrar
      </Button>
    </Form>
  </Card.Body>
</Card>
)
}
```

2.7 Criando AccessDenied

Vamos criar uma página para que servirá para que o usuário veja que está com acesso negado ou que o token expirou e precisa refazer o login, para isso, iremos criar um arquivo em **src/pages/AccessDenied/index.js**

```
import { Card, Col, Container, Row } from "react-bootstrap";
import { BsSignStopFill } from 'react-icons/bs'
import styles from './styles.module.scss';

export function AccessDenied(){
  return(
    <Row className={styles.container}>
      <Container>
        <Col xs={12} sm={9} md={6}>
          <Card className={styles.card}>
            <Card.Header className={styles.card__header}>
              Acesso Negado
            </Card.Header>
            <Card.Body>
              <Card.Text>
                <Row>
                  <Col>
                    <BsSignStopFill size={90} color="red" />
                  </Col>
                  <Col>
                    Você não tem permissão para acessar essa página ou o token está expirado
                  </Col>
                </Row>
              </Card.Text>
            </Card.Body>
          </Card>
        </Col>
      </Container>
    </Row>
  )
}
```

Sinta-se a vontade para criar e modificar o estilo da página.

2.8 Criando Protected Route

Vamos instalar o jwt-decode, para conseguirmos ler o nosso JWT

```
npm i jwt-decode
```

Agora iremos proteger nossas rotas, para isso vamos criar um arquivo em **src/pages/ProtectedRoute/index.js**

```
import { useEffect, useState } from "react";
import jwt_decode from 'jwt-decode';

export default function ProtectedRoute({ errorPage, targetPage }){
  var [page, setPage] = useState(<></>);

  function renderPage(){
    const token = sessionStorage.getItem('token');
    console.log(token)
    if(!token) {
      setPage(errorPage)
      return
    }

    const decodeToken = jwt_decode(token)
    const { exp } = decodeToken;

    if(exp+'000' - Date.now()){
      setPage(errorPage)
      return
    }
    setPage(targetPage)
  }

  useEffect(() => {
    renderPage()
  }, [])

  return page;
}
```

Por fim, basta importar nas rotas e testar, **mas isso iremos fazer só após integrar com a API**, senão iremos ficar sem token e não acessaremos as páginas

3 Back-end

3.1 Criando o projeto

Como vimos nas aulas anteriores, para criar o projeto em node, vamos começar com

```
npm init -y
```

e após concluir a criação, vamos instalar alguns pacotes

```
npm install express mongoose nodemon body-parser cors config
```

Vamos criar uma pasta chamada **src** e dentro dela vamos criar uma pasta chamada **controller**

```
class ArticleController {}  
  
module.exports = ArticleController;
```

Por enquanto ficará vazio só adicionando a exportação, mas logo mais iremos incrementá-la
Agora, vamos criar um arquivo **server.js** na raiz do projeto e colocar o código

```
const express = require('express');  
const routes = require('./src/routes');  
const cors = require('cors');  
  
const app = express();  
  
require('./startup/db')();  
  
require('./startup/routes')(app);  
  
app.use(cors({
```

```
    origin: '*'
  }));

  routes(app);

  const port = 8080;

  app.listen(port, () => console.log(`Acesse: http://localhost:${port}/`));
```

3.2 Configurando o banco de dados

Agora precisamos criar uma pasta **model**, dentro de **src** e criar um arquivo chamado **author.js**

```
const mongoose = require('mongoose');

const authorSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    minlength: 3
  },
  birth: {
    type: Date,
    required: true
  },
  createdAt: {
    type: Date,
    required: true
  },
  updatedAt: {
    type: Date,
    required: false
  },
  removedAt: {
    type: Date,
    required: false
  }
});
```



```
const Author = mongoose.model('Author',authorSchema);

exports.Author = Author;
exports.authorSchema = authorSchema;
```

Após isso, ainda dentro de model, criaremos um arquivo chamado **article.js**

```
const mongoose = require('mongoose');
const { authorSchema } = require('./author');

const Article = mongoose.model('Article',
  new mongoose.Schema({
    title: {
      type: String,
      required: true,
      minlength: 3
    },
    text: {
      type: String,
      required: true,
      minlength: 15
    },
    author: {
      type: authorSchema,
      required: true
    },
    likes: {
      type: Number,
      required: true
    },
    createdAt: {
      type: Date,
      required: true
    },
    updatedAt: {
      type: Date,
      required: false
    }
  })
);
```

```
    },
    removedAt: {
      type: Date,
      required: false
    },
  }
});

module.exports = Article
```

Após isso, vamos criar uma pasta **startup** na raiz do projeto, e um arquivo **db.js**

```
const mongoose = require('mongoose');
const config = require('config')

module.exports = function(){
  const db = config.get('db');
  mongoose.connect(db, { useNewUrlParser: true, useUnifiedTopology: true })
    .then(() => console.log(`connected to ${db}`));
}
```

Por fim, para terminar de configurar o banco, falta criar uma pasta chamada **config**, na pasta raiz, e criar o arquivo **default.json**

```
{
  "db": "mongodb://localhost:27017/Blog"
}
```

3.3 Configurando as Rotas

Dentro de **startup** vamos criar um novo arquivo chamado **routes.js**

```
const express = require('express');
const article = require('../src/routes/article');

module.exports = function(app) {
  app
    .use(express.json())
    .use('/api/article', article)
}
```

Agora, em **src** vamos criar uma pasta chamada **routes**, e um arquivo chamado **index.js**

```
const bodyParser = require('body-parser');
const article = require('./article');
const author = require('./author');

module.exports = (app) => {
  app.use(
    bodyParser.json(),
    article,
    author
  )
}
```

Agora, ainda em **src/routes** vamos criar o arquivo **article.js**

```
const express = require('express');
const ArticleController = require('../controller/articlesController');
const route = express.Router();

route
  .post('/api/article/', ArticleController.create)

module.exports = route;
```

Ainda em **src/routes** vamos criar um arquivo **author.js**

```
const express = require('express');
const AuthorController = require('../controller/authorController');
const route = express.Router();

route
  .get('/api/author', AuthorController.getAll)
  .post('/api/author', AuthorController.create)

module.exports = route;
```

3.4 Criando o controller

Para os autores, ainda não temos um controller, vamos então criar um arquivo chamado **authorController.js** dentro de **src/controller**

```
const { Author } = require("../model/author");
const User = require("../model/login");

class AuthorController{
  static async create(req, res){
    const { name, email, birth } = req.body;

    if(!name || !birth || !email)
      return res.status(400).send({ message: "os campos não podem estarem vazios " });

    if(name.length < 3)
      return res.status(400).send({ message: "o nome não pode ser menor que 3 caracteres" });

    if(email.length < 3)
      return res.status(400).send({ message: "Insira um e-mail válido" });

    if(!email.includes('@'))
      return res.status(400).send({ message: "Insira um e-mail válido" })

    const author = {
      name,
      email,
      birth,
      createdAt: Date.now(),
      updatedAt: Date.now(),
      removedAt: null,
    }

    try {
      await Author.create(author)
      return res.status(201).send({ message: "Autor cadastrado com sucesso" })
    } catch (error) {
      return res.status(500).send({ error: "Failed to get data" });
    }
  }
}
```

```
}

static async getAuthor(_id){
  try {
    const author = await User.findById({ _id })
    return author.author
  } catch (error) {
    throw error;
  }
}

module.exports = AuthorController;
```

E vamos testar o endpoint.

Body Cookies Headers (8) Test Results

Status: 201 Created

Pretty

Raw

Preview

Visualize

JSON

↺

```
1 {
2   "message": "Autor cadastrado com sucesso"
3 }
```

Com a configuração atual, o articleController ainda não possui nenhum método chamado create, então vamos criá-lo, em **src/controller/articleController.js**

```
static async create(req, res){
  const { title, text, authorid } = req.body;

  if(!title || !text || !authorid)
    return res.status(400).send({ message: "os campos não podem estarem vazios " });
```

```
if(title.length < 3)
  return res.status(400).send({ message: "o titulo não pode ser menor que 3 caracteres" });

if(text.length < 15)
  return res.status(400).send({ message: "o artigo não pode ser menor que 15 caracteres" });

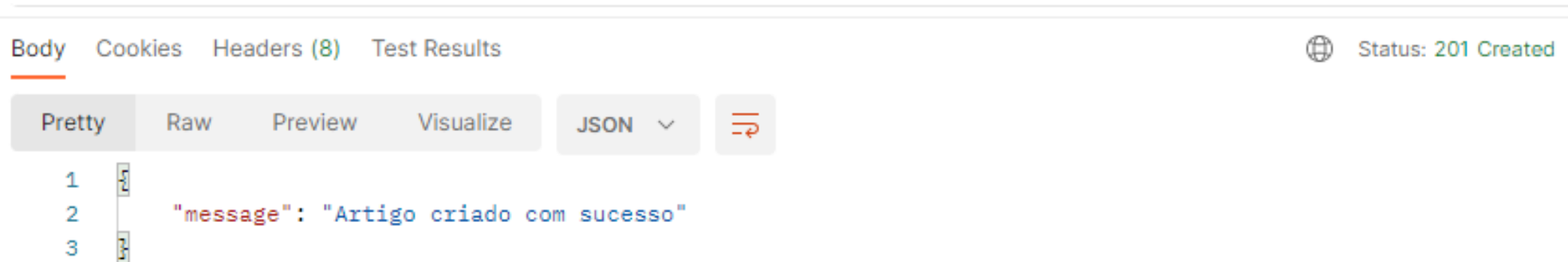
try {
  const author = await authorController.getAuthor(authorid);
  const article = {
    title,
    text,
    likes: 0,
    author,
    createdAt: Date.now(),
    updatedAt: Date.now(),
    removedAt: null,
  }
  await Article.create(article)
  return res.status(201).send({ message: "Artigo criado com sucesso" })
} catch (error) {
  ArticleController.createLog(error);
  return res.status(500).send({ error: "Falha ao salvar o artigo", data: error.message });
}
};
```

Agora podemos rodar o servidor e testar nossa rota, para isso vamos digitar **npm start**

Ao enviarmos um JSON com as informações necessárias

```
{
  "title": "Teste",
  "authorid": "64919a063057bc22fdd3f505",
  "text": "Teste com mais de 15 caracteres"
}
```

E temos o resultado a seguir:



Agora teste as validações para verificar se estão satisfazendo os if's.

3.5 Criando operação de curtir um artigo

Até o momento, os dados eram carregados com uma quantidade estática de curtidas, vamos criar agora como curtir o artigo

```
static async likeArticle(req, res){
  const { id } = req.params;

  if(!id) return res.status(400).send({ message: "No id provider" })

  try {
    const article = await Article.findById(id);
    await Article.findByIdAndUpdate({_id: id}, {likes: ++article.likes})
    return res.status(200).send();
  } catch (error) {
    ArticleController.createLog(error);
    return res.status(500).send({ error: "Falha ao curtir", data: error.message })
  }
}
```

3.6 Registrando Usuário

Para iniciar o registro de usuário, precisamos conectar com o banco de dados, para isso iremos criar um arquivo em **src/models** com o nome de **User** e colocar o código

```
const mongoose = require('mongoose');
const { authorSchema } = require('./author');

const User = mongoose.model('User',
  new mongoose.Schema({
    author: {
      type: authorSchema,
      required: true
    },
    login: {
      type: String,
      required: true,
      minlength: 3
    },
    password: {
      type: String,
      required: true,
      minlength: 6
    },
    email: {
      type: String,
      required: true,
      minlength: 6
    },
    createdAt: {
      type: Date,
      required: true
    },
    updatedAt: {
      type: Date,
      required: false
    },
    removedAt: {
```



```
        type: Date,  
        required: false  
      },  
    })  
  )  
  
  module.exports = User
```

Agora iremos criar um método para criar usuário, para isso precisamos criar um arquivo chamado **UserController** dentro da pasta **controller**

```
const User = require('../model/login');  
const jwt = require('jsonwebtoken');  
const { Author } = require('../model/author');  
require('dotenv').config();  
const CryptoJS = require("crypto-js");  
  
class AuthController{  
  static async register(req, res){  
  
    var bytes = CryptoJS.AES.decrypt(req.body.jsonCrypt, process.env.SECRET);  
    const decryptd = bytes.toString(CryptoJS.enc.Utf8);  
    const json = JSON.parse(decryptd);  
  
    const { name, birth, email, password, confirmPassword } = json;  
  
    if(!name)  
      return res.status(400).json({ message: "O nome é obrigatório" });  
  
    if(!email)  
      return res.status(400).json({ message: "O e-mail é obrigatório" });  
  
    if(!password)  
      return res.status(400).json({ message: "A senha é obrigatória" });  
  
    if(password !== confirmPassword)  
      return res.status(400).json({ message: "As senhas não conferem" });  
  
    const userExist = await User.findOne({ email: email });
```

```
if(userExist)
  return res.status(422).json({ message: "insira outro e-mail" });

const passwordCrypt = CryptoJS.AES.encrypt(password, process.env.SECRET).toString();

const author = new Author({
  name,
  email,
  birth,
  createdAt: Date.now(),
  updatedAt: Date.now(),
  removedAt: null,
})

const user = new User({
  login: email,
  author,
  email,
  password: passwordCrypt,
  createdAt: Date.now(),
  updatedAt: Date.now(),
  removedAt: null,
});

try {
  await User.create(user);
  res.status(201).send({ message: "Usuário cadastrado com sucesso" });
} catch (error) {
  return res.status(500).send({ message: "Something failed", data: error.message })
}

}

module.exports = AuthController;
```

Vamos também criar um arquivo de rotas para o controller

```
const express = require('express');
const AuthController = require('../controller/authController');
const route = express.Router();

route
  .post('/api/register', AuthController.register)

module.exports = route;
```

E importa-lo no **routes/index**

```
const bodyParser = require('body-parser');
const article = require('./article');
const author = require('./author');
const user = require('./user');

module.exports = (app) => {
  app.use(
    bodyParser.json(),
    article,
    author,
    user
  )
}
```

Para testarmos a nossa requisição basta inserir o código no endpoint <http://localhost:8080/api/register>

```
{
  "name": "Teste",
  "lastname": "teste",
  "birth": "01/01/1996",
  "login": "Teste",
  "email": "teste@teste.com",
  "password": "teste",
  "confirmPassword": "teste"
```

```
}
```

E o resultado será

Body Cookies Headers (8) Test Results

Status: 201 Created

Pretty

Raw

Preview

Visualize

JSON



1

2

3

```
"message": "Usuário cadastrado com sucesso"
```

3.7 Login do usuário

Para realizarmos o login, vamos aproveitar o o **AuthController** e adicionar o método de login.

4 Integrando o front-end com API

4.1 Exibindo os artigos

O primeiro local que vamos integrar será a página dos artigos, para isso vamos fazer a chamada da API e um **console.log** nos resultados

```
async function getPosts(){  
  const res = await axios.get('http://localhost:8080/api/article')  
  console.log(res);  
}
```

E nossa resposta no console.log será semelhante a imagem abaixo, podendo ter menos ou mais elementos dentro do array, caso o array retorne vazio, é interessante fazer pelo menos uma inserção na API

[index.js:16](#)

```

{data: Array(1), status: 200, statusText: 'OK', headers: AxiosHeaders, confi
g: {...}, ...} 1
  ▶ config: {transitional: {...}, adapter: Array(2), transformRequest: Array(1),
  ▼ data: Array(1)
    ▼ 0:
      ▼ author:
        birth: "1995-01-01T02:00:00.000Z"
        createdAt: "2023-06-27T13:00:36.142Z"
        lastname: "Teste"
        name: "teste"
        removedAt: null
        updatedAt: "2023-06-27T13:00:36.142Z"
        __v: 0
        _id: "649add74d661381164caeb69"
        ▶ [[Prototype]]: Object
        createdAt: "2023-06-27T13:06:28.043Z"
        likes: 0
        removedAt: null
        text: "Teste com mais de 15 caracteres"
        title: "teste"
        updatedAt: "2023-06-27T13:06:28.043Z"
        __v: 0
        _id: "649aded4d661381164caeb6c"
        ▶ [[Prototype]]: Object
        length: 1
        ▶ [[Prototype]]: Array(0)
      ▶ headers: AxiosHeaders {content-length: '422', content-type: 'application/js
      ▶ request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout:
        status: 200
        statusText: "OK"
      ▶ [[Prototype]]: Object

```

Agora que vimos que nossos dados estão dentro de **res.data**, então colocamos o **res.data dentro do setArtigos** e podemos apagar os artigos que antes estavam mocados, ficando assim

```
async function getPosts(){
  const res = await axios.get('http://localhost:8080/api/article')
  setArtigos(res.data)
}
```

4.2 Curtindo um artigo

Até o momento, nosso botão de curtir não estava fazendo nada, vamos dar funcionalidade ao botão, vamos colocar o código

```
async function handleClick(id){
  await axios.post(`http://localhost:8080/api/article/like/${id}`)
  getPosts();
}
```

e no botão colocar

```
onClick={() => handleClick(artigo._id)}
```

4.3 Criando um autor

Para criar um autor, antes precisamos criptografar o JSON para enviar.

Para criptografar, será semelhante ao processo que fizemos no back-end

Agora vamos criar um novo autor, para isso, vamos no **CardRegister** e implementar a função de criar um novo autor. Vamos **substituir a função handleSubmit**

```
async function handleSubmit(e){
  e.preventDefault();
  if(!formValid()) return

  const json = {
    name, email, birth, password, confirmPassword
  }
  const jsonCrypt = CryptoJS.AES.encrypt(JSON.stringify(json), SECRET).toString();
```

```
try {
  var res = await axios.post('http://localhost:8080/api/author',{
    jsonCrypt
  })

  setMessage(res.data.message);
  setVariant('success');
  setShow(true);
  setName('');
  setEmail('');
  setPassword('');
  setConfirmPass('');
} catch (error) {
  console.log(error);
}
```

e importar no projeto

```
import { SECRET } from "../..env";
import CryptoJS from 'crypto-js';
```

Agora falta criar o env com uma senha secreta e **não podemos enviar esse arquivo para o git**, será um

```
export const SECRET='AFAAFipjafijAPFJpjpasjfpJA3254Aas34saf2'
```

Por fim vamos testar, e nossa resposta deverá ser status 201 salvando no banco de dados


```
  _id: ObjectId('649ec9bd9963c3f402c2dc78')
  author: Object
    name: "aaasadsasd asd"
    email: "asdsa@teste.com"
    birth: 1995-01-01T00:00:00.000+00:00
    createdAt: 2023-06-30T12:25:33.789+00:00
    updatedAt: 2023-06-30T12:25:33.789+00:00
    removedAt: null
    _id: ObjectId('649ec9bd9963c3f402c2dc77')
  login: "asdsa@teste.com"
  password: "U2FsdGVkX1/WOJ5uU2PWp/DQjoJtyMIVPXzeYkEuG7I="
  email: "asdsa@teste.com"
  createdAt: 2023-06-30T12:25:33.795+00:00
  updatedAt: 2023-06-30T12:25:33.795+00:00
  removedAt: null
  __v: 0
```

Observe que quem conseguir acessar o banco, somente irá ver uma senha criptografada, faça o teste

4.4 Login do usuário

Semelhante a registrar, precisamos criptografar o JSON para enviar a requisição.

Na API precisamos criar uma rota para o login

```
static async login(req, res){
  var bytes = CryptoJS.AES.decrypt(req.body.jsonCrypt, process.env.SECRET);
  const decryptd = bytes.toString(CryptoJS.enc.Utf8);
  const json = JSON.parse(decryptd);

  const { email, password } = json;

  if(!email)
    return res.status(422).json({ message: "O e-mail é obrigatório" });
```

```
if(!password)
  return res.status(422).json({ message: "A senha é obrigatória" });

const user = await User.findOne({ email: email });
if(!user)
  return res.status(422).json({ message: "Usuário e/ou senha inválido" });

if(!await bcrypt.compare(password, user.password))
  return res.status(422).send({ message: "Usuário e/ou senha inválido" })

try {
  const secret = process.env.SECRET
  const token = jwt.sign(
    {
      id: user._id,
    },
    secret,
    {
      expiresIn: '2 days'
    }
  );

  return res.status(200).send({token: token})
} catch (error) {
  return res.status(500).send({ message: "Something failed", data: error.message })
}
```

No front-end, vamos alterar a função **handleSubmit** para a seguinte.

```
async function handleSubmit(e){
  e.preventDefault();
  if(!formValid()) return

  const json = {
```

```
    email, password
  }
  try {
    const jsonCrypt = CryptoJS.AES.encrypt(JSON.stringify(json), SECRET).toString();
    var res = await axios.post('http://localhost:8080/api/login',{
      jsonCrypt
    })
    sessionStorage.setItem('token', res.data.token);
    navigate('/home')
  } catch (error) {

    setMessage('Erro ao se conectar');
    setShow(true);
    setVariant('danger');
  }
}
```

4.5 Protegendo as Rotas

No final da criação do front-end, criamos a página de **ProtectedRoute**, mas não usamos pois estávamos sem token, agora integrando a API com o front-end, vamos de fato proteger as rotas

Para isso, no arquivo **App.js**

```
import { Route, Routes } from 'react-router-dom';
import HomePage from './pages/home';
import AddPostPage from './pages/AddPost';
import LoginPage from './pages/Login';
import RegisterPage from './pages/Register';
import { AlertProvider } from './context/alert';
import './App.css';
import ProtectedRoute from './pages/ProtectedRoute';
import { AccessDenied } from './pages/AccessDenied';
import NavBar from './components/navbar';
import NotFoundPage from './pages/NotFoundPage';

function App() {
  return (
```

```
<>
  <AlertProvider>
    <Routes>
      <Route path="/" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/main" element={
        <ProtectedRoute
          errorPage={<AccessDenied />}
          targetPage={<NavBar />}
        />
      } />
      <Route path="/" element={<HomePage />} />
      <Route path="/add" element={<AddPostPage />} />
    </Routes>
    <Route path="*" element={<NotFoundPage />} />
  </Routes>
</AlertProvider>
</>
);
}
```

```
export default App;
```

4.6 Criando AccessDenied

Vamos criar o nosso accessDenied agora, é uma página em que mostrará que o usuário não está autorizado a entrar, para isso, iremos criar um arquivo em **src/pages/AccessDenied/index.js**

```
import { Card, Col, Container, Row } from "react-bootstrap";
import { BsSignStopFill } from 'react-icons/bs'
import styles from './styles.module.scss';

export function AccessDenied(){
  return(
    <Row className={styles.container}>
      <Container>
        <Col xs={12} sm={9} md={6}>
          <Card className={styles.card}>
```

```
        <Card.Header className={styles.card__header}>
          Acesso Negado
        </Card.Header>
        <Card.Body>
          <Card.Text>
            <Row>
              <Col>
                <BsSignStopFill size={90} color="red" />
              </Col>
              <Col>
                Você não tem permissão para acessar essa página ou o token está expirado
              </Col>
            </Row>
          </Card.Text>
        </Card.Body>
      </Card>
    </Col>
  </Container>
</Row>
)
}
```

4.7 Criando um artigo

Agora iremos integrar a operação de criar um artigo. No arquivo **src/components/Formulario/index.js**

```
import { useContext, useState } from "react";
import axios from "axios";
import jwt_decode from 'jwt-decode';
import {
  Button,
  Col,
  Container,
  Form,
  Row
} from "react-bootstrap";
```

```
import { AlertContext } from "../../context/alert";
import styles from './styles.module.scss';

export default function Formulario(){
  const { setMessage, setShow, setVariant } = useContext(AlertContext);

  var [title, setTitle] = useState('');
  var [text, setText] = useState('');

  async function handleSubmit(e){
    e.preventDefault();
    try {
      const token = sessionStorage.getItem('token');
      const decodeToken = jwt_decode(token)
      const { id } = decodeToken;

      const res = await axios.post('http://localhost:8080/api/article', {
        authorid: id, title, text
      });

      setMessage(res.data.message);
      setShow(true);
      setVariant('success');
      setTitle('');
      setText('');
    } catch (error) {
      console.log(error);
      setMessage("Erro ao inserir o artigo, reveja as informações e tente novamente");
      setShow(true);
      setVariant('danger');
    }
  }

  return(
    <Container>
      <Row>
        <Col>
          <Form onSubmit={handleSubmit} className={styles.form}>
```

```

        <Form.Text className={styles.form__title}>Digite Aqui seu Artigo</Form.Text>
        <Form.Control
          placeholder="Titulo"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
        />
        <Form.Control
          as='textarea'
          placeholder="Texto"
          rows={5}
          value={text}
          onChange={(e) => setText(e.target.value)}
        />
        <Col xs={12} sm={9} md={6} className={styles.form__div}>
          <Button type="submit" className={styles.form__div__button}>Salvar</Button>
        </Col>
      </Form>
    </Col>
  </Row>
</Container>
)
}

```

4.8 Paginação

Agora iremos implantar opção de paginação, para isso na API, no arquivo **src/controller/articlesController.js**, vamos modificar o método `getAll`

```

const Article = require('../model/article');
const authorController = require('../authorController');
const fs = require('fs');
const path = require('path');

class ArticleController {
  static createLog(error){
    const timestamp = Date.now();
    const archivePath = path.resolve(__dirname, '..', `logs-${timestamp}.txt`);
    const errorString = JSON.stringify(error.message)
    fs.writeFile(archivePath, errorString, function(err, result) {
      if(err) console.log(err)
    })
  }
}

```

```
    })
  }

  static async getAll(req, res){
    let page = req.params.page;
    let limit = 5;
    let skip = limit * (page - 1);
    try {
      const articles = await Article.find().skip(skip).limit(limit);
      return res.status(200).send(articles);
    } catch (error) {
      ArticleController.createLog(error);
      return res.status(500).send({ message: "Falha ao carregar os Artigos"})
    }
  };

  static async create(req, res){
    const { title, text, authorid } = req.body;

    if(!title || !text || !authorid)
      return res.status(400).send({ message: "os campos não podem estarem vazios " });

    if(title.length < 3)
      return res.status(400).send({ message: "o titulo não pode ser menor que 3 caracteres" });

    if(text.length < 15)
      return res.status(400).send({ message: "o artigo não pode ser menor que 15 caracteres" });

    if(authorid.length < 3)
      return res.status(400).send({ message: "O autor não pode ser menor que 3 caracteres" })

    try {
      const author = await authorController.getAuthor(authorid);
      const article = {
        title,
        text,
        likes: 0,
        author,
```



```
        createdAt: Date.now(),
        updatedAt: Date.now(),
        removedAt: null,
    }
    await Article.create(article)
    return res.status(201).send({ message: "Artigo criado com sucesso" })
} catch (error) {
    ArticleController.createLog(error);
    return res.status(500).send({ error: "Falha ao salvar o artigo", data: error.message });
}
};

static async likeArticle(req, res){
    const { id } = req.params;

    if(!id) return res.status(400).send({ message: "No id provider" })

    try {
        const article = await Article.findById(id);
        await Article.findByIdAndUpdate({_id: id}, {likes: ++article.likes})
        return res.status(200).send();
    } catch (error) {
        ArticleController.createLog(error);
        return res.status(500).send({ error: "Falha ao curtir", data: error.message })
    }
}

module.exports = ArticleController;
```

Agora no arquivo de rotas em **src/routes/article.js**

```
const express = require('express');
const ArticleController = require('../controller/articlesController');
const route = express.Router();

route
```

```
.get('/api/article/:page', ArticleController.getAll)
.post('/api/article/', ArticleController.create)
.post('/api/article/like/:id', ArticleController.likeArticle)

module.exports = route;
```

Por fim no react em **src/components/Post/index.js**

```
import { useEffect, useState } from 'react';
import {
  Button,
  Card,
  Container
} from 'react-bootstrap'
import { AiOutlineLike } from 'react-icons/ai'
import styles from './styles.module.scss';
import axios from 'axios';

export default function Post(){
  var [page, setPage] = useState(1);
  var [artigos, setArtigos] = useState([]);

  async function getPosts(){
    const res = await axios.get(`http://localhost:8080/api/article/${page}`)
    setArtigos(res.data)
  }

  useEffect(() => {
    getPosts();
  }, [page])

  async function handleClick(id){
    await axios.post(`http://localhost:8080/api/article/like/${id}`)
    getPosts();
  }
}
```

```
function handleUp(){
  if(artigos.length===5){
    setPage(++page)
  }
}

function handleDown(){
  if(page>1){
    setPage(--page)
  }
}

const RenderPosts = () => {
  return artigos.map((artigo) => {
    return(
      <Card key={artigo._id} className={styles.card} >
        <Card.Title className={styles.card__title}>
          {artigo.title}
        </Card.Title>
        <Card.Body className={styles.card__body}>
          <Card.Text className={styles.card__body__article}>{artigo.text}</Card.Text>
          <div className='d-flex align-items-center '>
            {artigo.likes}<Button variant='light' onClick={() => handleClick(artigo._id)}><AiOutlineLike /></Button>
          </div>
        </Card.Body>
      </Card>
    )
  })
}

return(
  <Container>
    <RenderPosts />
    <Button onClick={handleDown}>-</Button>
    {page}
    <Button onClick={handleUp}>+</Button>
  </Container>
)
```

4.9 Agora é com você

Desde o começo do curso vimos diversas coisas, agora, com base em todo o conhecimento adquirido faça os desafios proposto.

- Desafio 1: Faça a parte dos comentários para o artigo
- Desafio 2: Corrija a curtida dos artigos que um usuário pode curtir diversas vezes
- Desafio 3: Implemente a opção de curtir um comentário
- Desafio 4: Implemente uma tradução no site (Obs: não traduza os arquivos do banco, somente o front-end)