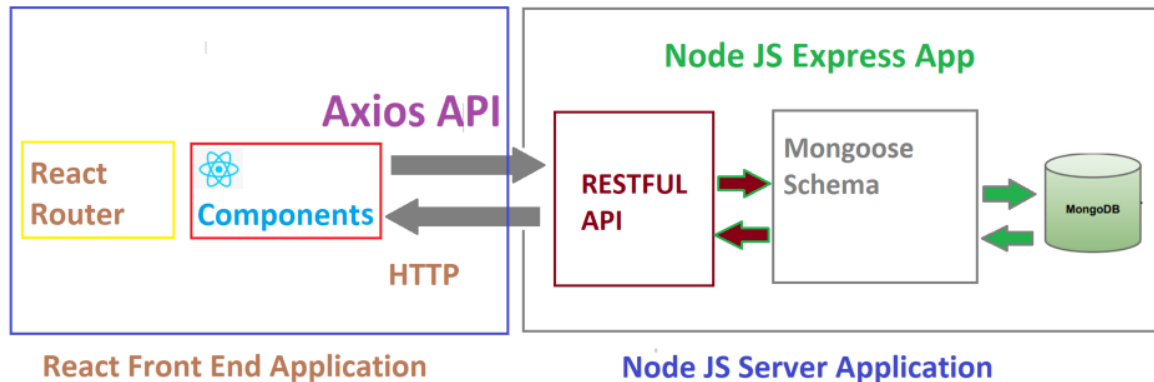




Projeto Full-Stack MERN Livraria

Arquitetura de um projeto MERN



1. Configuração do servidor com Express.js e Node.js

Este roteiro foi feito com o objetivo de destacar a configuração MERN. O objetivo é desenvolver um projeto simples com a melhor estrutura possível para que você possa usá-lo como padrão e elevar seus projetos de stack MERN para atender aos padrões do mercado.

Para começar nosso tutorial de stack MERN, mostraremos como configurar um servidor com Express.js e Node.js.

inicialização do pacote npm

Crie uma pasta para o projeto chamada `livraria-mern`. Dentro desta pasta crie mais 2 pastas. Uma pasta chamada `server`, onde será configurado o servidor de back-end e uma pasta chamada `client` onde estará o front-end feito em React.

Entre na pasta `server` pelo terminal e execute `$ npm init`. Em seguida, ele fará algumas perguntas sobre o nome do pacote, versão, ponto de entrada e muito mais.



Pressione **Enter** se quiser manter o padrão. Depois disso, você obterá algo assim:

```
nurislam@Nurs-MacBook-Pro ~/HackWay/MyGithub/MERN_A_to_Z master npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (mern_a_to_z)
version: (1.0.0)
description:
entry point: (index.js) app.js
test command:
git repository: (https://github.com/nurislam03/MERN_A_to_Z.git)
keywords:
author: Nur Islam
license: (ISC) MIT
About to write to /Users/nurislam/HackWay/MyGithub/MERN_A_to_Z/package.json:

{
  "name": "mern_a_to_z",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/nurislam03/MERN_A_to_Z.git"
  },
  "author": "Nur Islam",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/nurislam03/MERN_A_to_Z/issues"
  },
  "homepage": "https://github.com/nurislam03/MERN_A_to_Z#readme"
}

Is this OK? (yes) yes
```



Selecione `yes` e você está pronto para começar. Isso criará um arquivo chamado `package.json`.

Instalando as dependências

Em seguida, adicionaremos algumas dependências com o comando `$ npm i express mongoose body-parser config`. Digite ou copie o comando acima e pressione o botão Enter. Você verá algo assim:

```
nurislam@Nurs-MacBook-Pro: ~/HackWay/MyGithub/MERN_A_to_Z [master] $ npm i express mongoose body-parser bcryptjs validation
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN mern_a_to_z@1.0.0 No description

+ bcryptjs@2.4.3
+ body-parser@1.19.0
+ express@4.17.1
+ mongoose@5.5.15
+ validation@0.0.1
added 74 packages from 57 contributors and audited 199 packages in 6.55s
found 0 vulnerabilities

nurislam@Nurs-MacBook-Pro: ~/HackWay/MyGithub/MERN_A_to_Z [master]
```

Aqui está o que obtemos no código acima:

- `body-parser`: Permite obter os dados ao longo da requisição, é um módulo capaz de converter o body da requisição para vários formatos. Um desses formatos é json, exatamente o que queremos;
- `express`: é o nosso framework principal;
- `mongoose`: É usado para conectar e interagir com o MongoDB;
- `config`: Isso permite que você defina parâmetros padrão para seu aplicativo.

Agora, adicionaremos o nodemon como uma dependência de `dev`. Se você não quiser adicionar isso, pode ignorá-lo - é opcional. Instale-o com `$ npm i -D nodemon`. Para usar o nodemon, adicione `"app": "nodemon app.js"` à sua tag de scripts no arquivo `package.json`.



Nodemon é um utilitário que monitorará quaisquer alterações em seu código e reiniciará automaticamente seu servidor. O `app.js` é o ponto de entrada para o aplicativo. Também é importante definir um script de início aqui com `"start": "node app.js"`. Isso definirá o script de inicialização do aplicativo.

Depois disso, seu `package.json` deve ficar assim:

```
{
  "name": "mern_a_to_z_server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node app.js",
    "app": "nodemon app.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.1",
    "config": "^3.3.8",
    "express": "^4.18.2",
    "mongoose": "^6.8.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

Definindo o ponto de entrada

Agora, crie um arquivo chamado `app.js` para nosso ponto de entrada.

`app.js`

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

const app = express();

app.get('/', (req, res) => res.send('Hello world!'));
```



```
const port = process.env.PORT || 8082;

app.listen(port, () => console.log(`Server running on port
${port}`));
```

Depois disso, execute o comando `$ node app`. Você verá `Server running on port 8082`. Você também pode verificar no navegador abrindo o navegador e digitando `http://localhost:8082`.

Neste ponto, se mudarmos alguma coisa, precisaremos reiniciar o servidor manualmente. Mas, se configurarmos o nodemon, não precisaremos reiniciá-lo todas as vezes. O Nodemon observará se houver alguma alteração e reiniciará o servidor automaticamente.

```
{

  "name": "server",

  "version": "1.0.0",

  "description": "",

  "main": "app.js",

  "scripts": {

    "start": "node app.js",

    "app": "nodemon app.js",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "",
```



```
"license": "ISC",

"dependencies": {

  "body-parser": "^1.20.1",

  "config": "^3.3.9",

  "express": "^4.18.2",

  "mongoose": "^6.9.0"

},

"devDependencies": {

  "nodemon": "^2.0.20"

}

}
```

Agora, você pode executar seu projeto usando o comando `$ npm run app`. Se você receber algum erro neste ponto, execute os comandos abaixo:

```
$ npm install
```

```
$ npm run app
```

Você verá as seguintes alterações em seu terminal se tudo der certo:

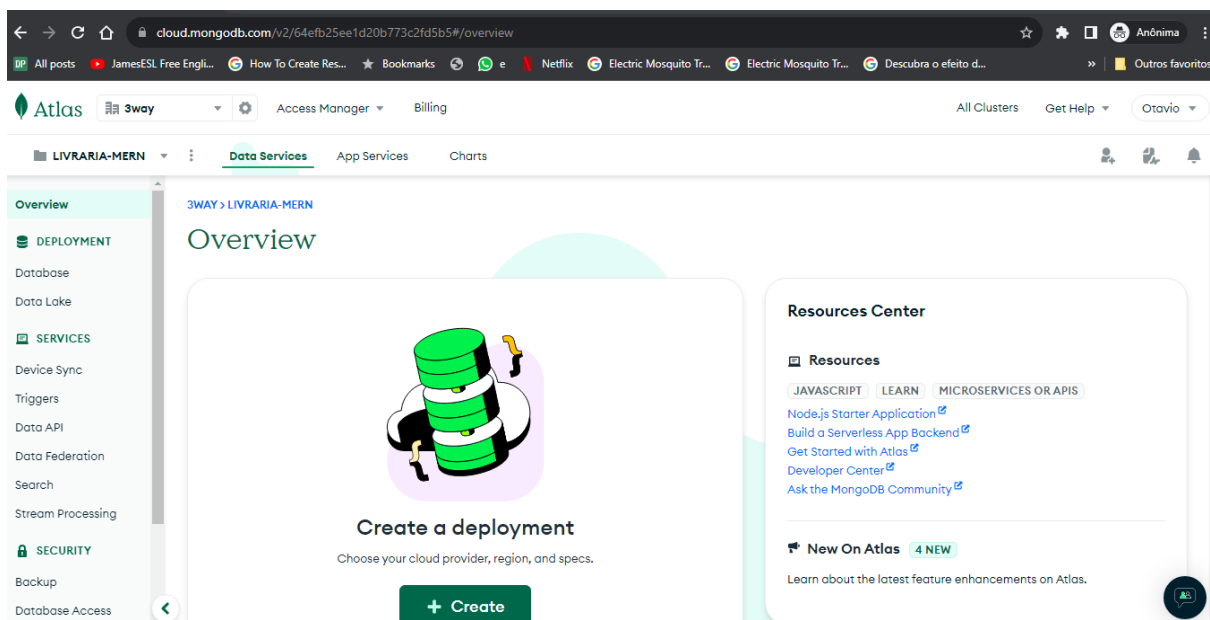


```
* nurislam@Nurs-MacBook-Pro ~/HackWay/MyGithub/MERN_A_to_Z master npm run app
> mern_a_to_z@1.0.0 app /Users/nurislam/HackWay/MyGithub/MERN_A_to_Z
> nodemon app.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Server running on port 8082
```

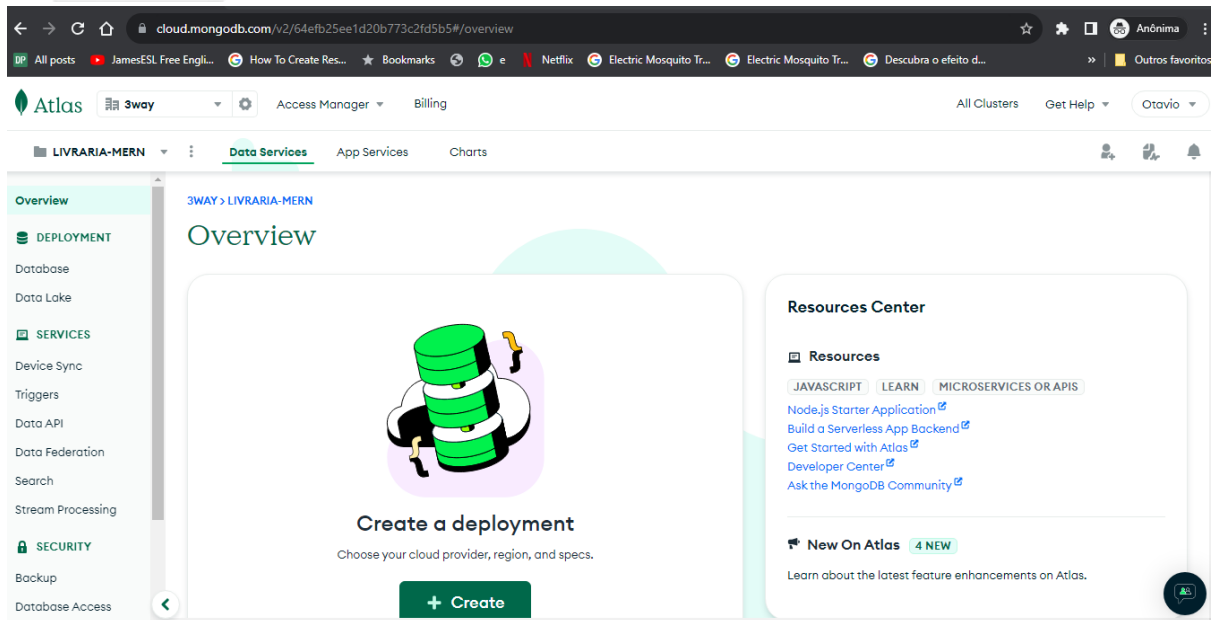
2. Gerenciamento de banco de dados com MongoDB

Agora, é hora de trabalhar na configuração do nosso banco de dados MERN com o MongoDB. Para simplificar, usaremos o MongoDB Atlas. Primeiro, crie uma conta [aqui](#). Depois de criar uma conta, você verá algo assim:

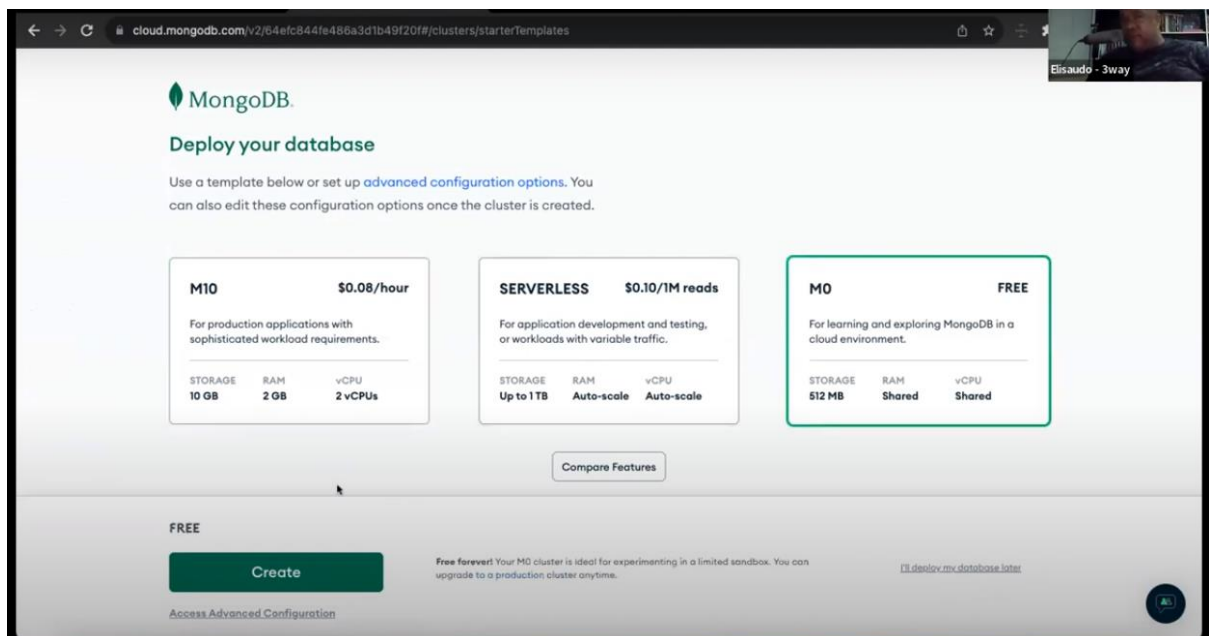


Clique na seção **Project 0** (canto superior esquerdo) e você verá um botão para Criar um novo projeto. Crie um projeto e selecione o projeto. Agora, clique no botão **Create** no projeto que você criou. Ele vai te mostrar todas as informações.

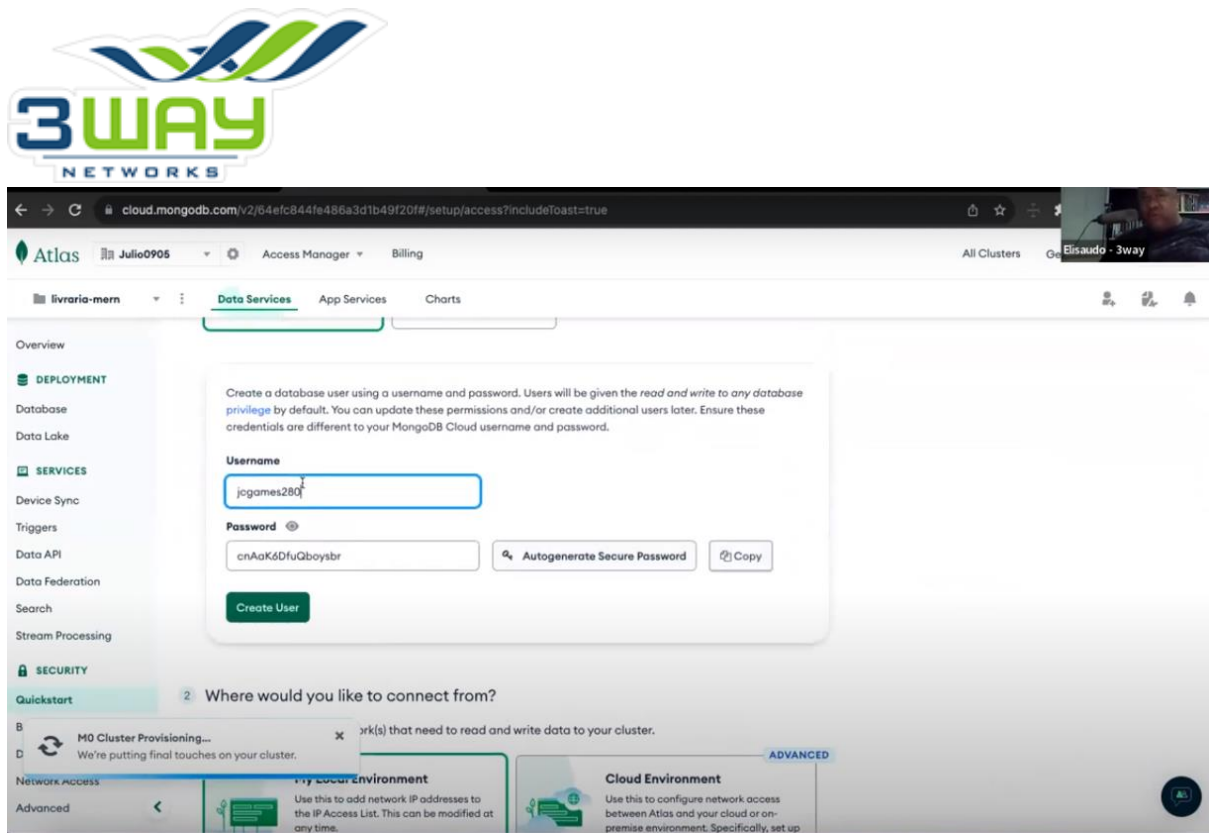
Na parte inferior, você verá uma seção chamada **Create Deployment**, clique nela.



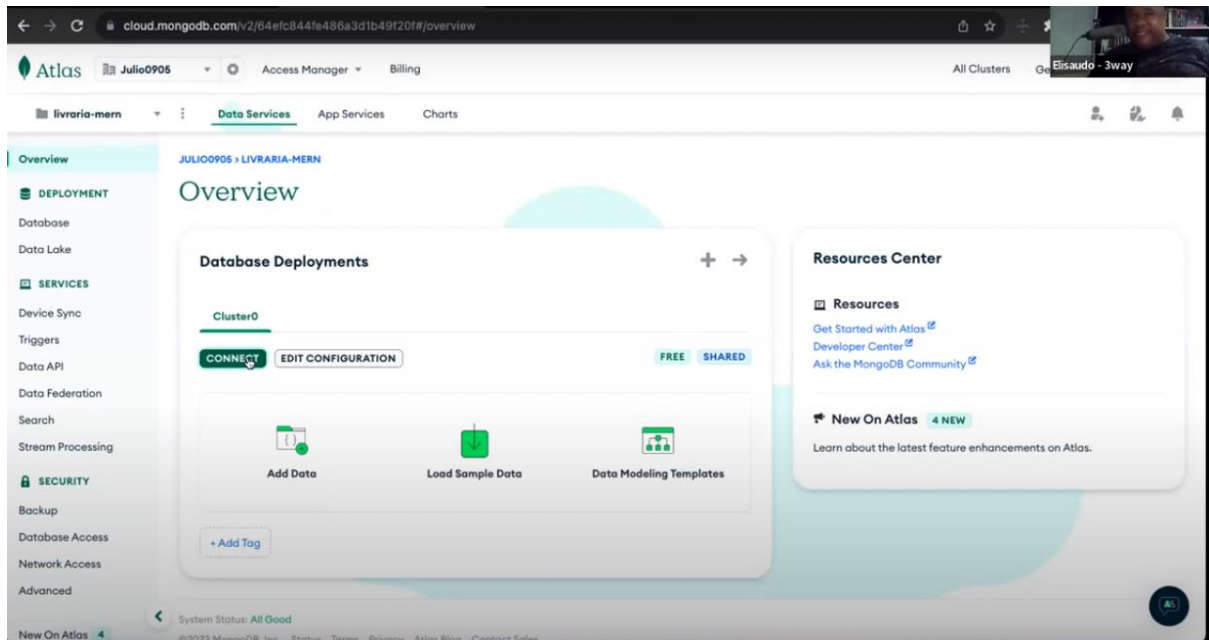
Selecione a opção FREE e clique em create.



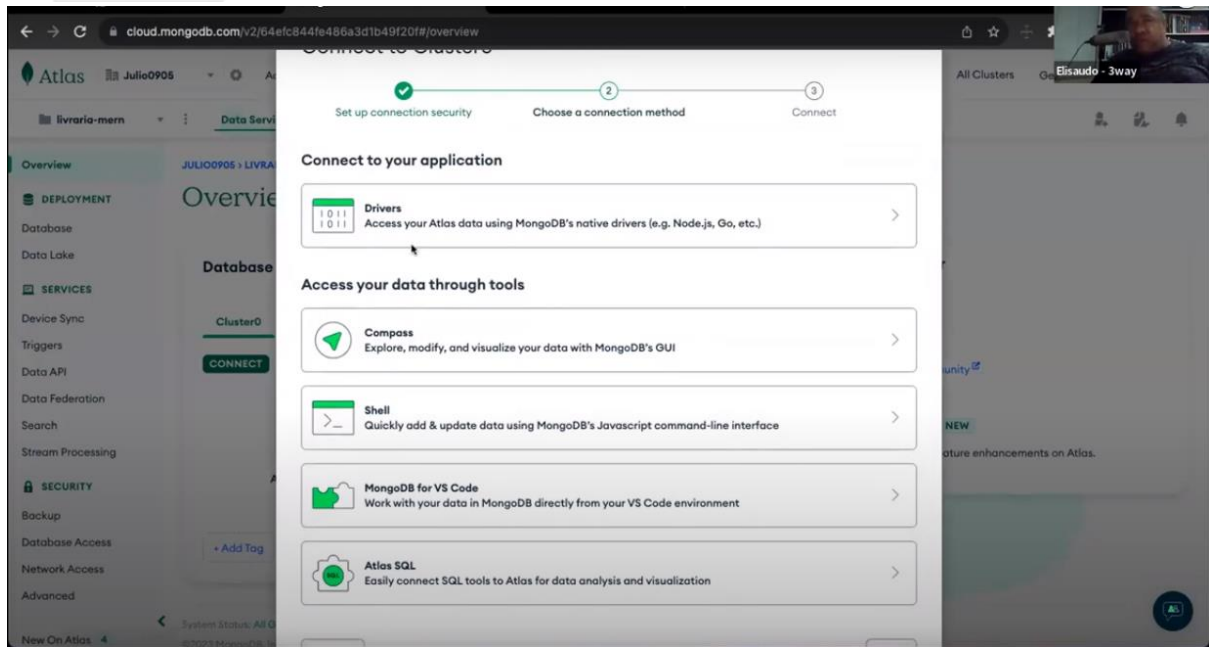
Aparecerá a tela para preencher o formulário de nome de usuário e senha para seu banco de dados. Tecle um **UserName and Password**, **Ad My Current IP** e selecione **Cloud Environment** e ao final tecle **Finish and Close**:



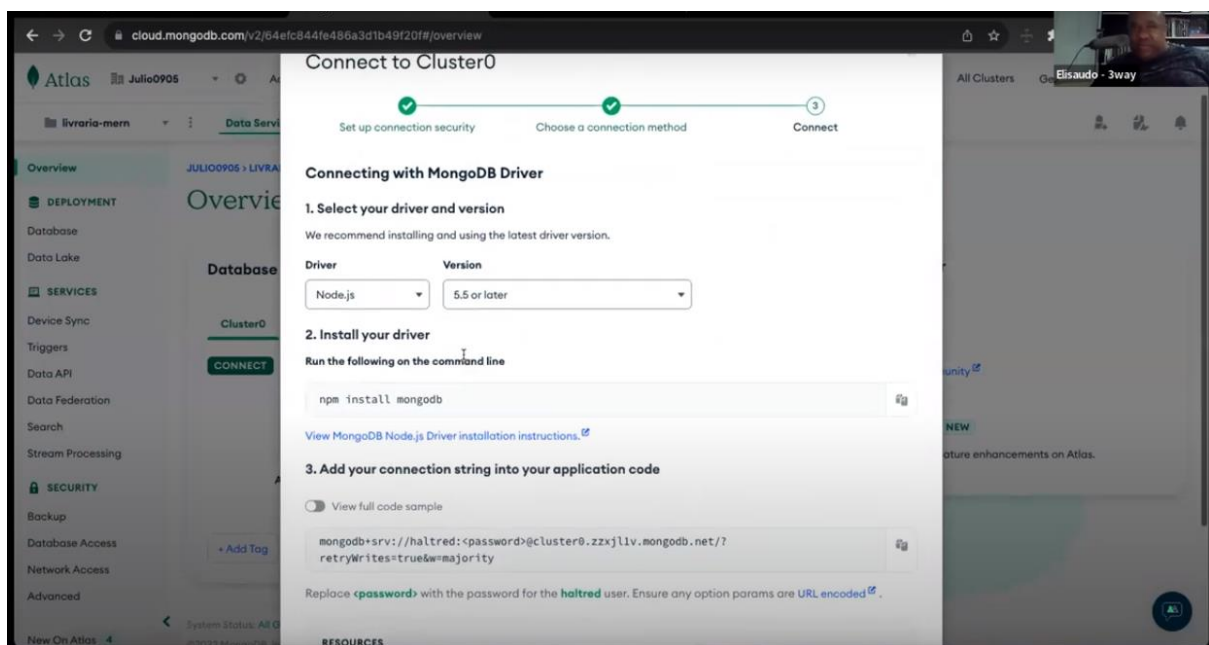
Na tela que aparece clique em **Connect**. Na aba **Database Deployment**.



Aparecerá a tela abaixo. Selecione **Connect to your application**:



Agora, você obterá seu link de banco de dados, que usaremos em nossa próxima etapa:



Adicionando o banco de dados ao nosso projeto



Nosso banco de dados está pronto e precisamos adicioná-lo ao nosso projeto. Dentro da pasta `server`, crie outra pasta chamada `config` e crie dois arquivos chamados `default.json` e `db.js`.

Adicione o seguinte código:

`default.json`

```
{  
  
  "mongoURI":  
  
    "mongodb+srv://mern123:<password>@mernatoz-  
9kdpd.mongodb.net/test?retryWrites=true&w=majority"  
  
}  
  
/* Substitua <password> pela senha do seu banco de dados*/
```

`db.js`

```
const mongoose = require('mongoose');  
  
const config = require('config');  
  
const db = config.get('mongoURI');  
  
const connectDB = async () => {  
  
  try {  
  
    mongoose.set('strictQuery', true);  
  
    await mongoose.connect(db, {
```



```
        useUrlParser: true,

    });

    console.log('MongoDB está conectado...');

    } catch (err) {

        console.error(err.message);

        process.exit(1);

    }

};

module.exports = connectDB;
```

<https://www.mongodb.com/community/forums/t/deprecationwarning-mongoose-the-strictquery/209637>

<https://mongoosejs.com/docs/5.x/docs/deprecations.html>

Precisamos de uma pequena alteração em nosso arquivo `app.js` para conectar ao banco de dados. Atualize seu `app.js` com isto:

`app.js`

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

const connectDB = require('./config/db');

const app = express();

// Conecta o banco de dados
```



```
connectDB();

app.get('/', (req, res) => res.send('Hello world!'));

const port = process.env.PORT || 8082;

app.listen(port, () => console.log(`Server running on port ${port}`));
```

Agora, você pode executar o projeto usando o comando `$ npm run app`. Você deve ver o seguinte:

```
* nurislam@Nurs-MacBook-Pro ~/HackWay/MyGithub/MERN_A_to_Z master npm run app
> mern_a_to_z@1.0.0 app /Users/nurislam/HackWay/MyGithub/MERN_A_to_Z
> nodemon app.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Server running on port 8082
MongoDB is Connected...
```

Ótimo! Até agora, estamos no caminho certo e nosso banco de dados está conectado com sucesso. Agora é hora de concluir a **configuração da rota** e, depois disso, veremos como **criar APIs RESTful**.

Construindo APIs RESTful com a stack MERN

Para começar, crie uma pasta chamada `routes`. Nele, crie outra pasta chamada `api`, que irá conter todas as nossas APIs. Dentro da pasta `api`, crie um arquivo



chamado `books.js`. Vamos criar algumas APIs aqui para mostrar como funciona daqui a pouco.

Agora, atualize seu `books.js` com o seguinte código:

book.js

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

const router = express.Router();

//Carrega o Book model

const Book = require('../models/Book');

//GET api/books/test

//Rota para testar livros

router.get('/test', (req, res) => res.send('Testando rota
book'));

//GET api/books

//Pegar todos os livros

router.get('/', (req, res) => {

  Book.find()

    .then(books => res.json(books))

    .catch(err => res.status(404).json({nobooksfound:
'Nenhum livro encontrado'}));
```



```
});

//@route GET api/books/:id

//Pegar único livro por id

router.get('/:id', (req, res) => {

    Book.findById(req.params.id)

        .then(book => res.json(book))

        .catch(err => res.status(404).json({ nobooksfound:
'Nenhum livro encontrado'}));

});

//POST api/books

//adicionar/salvar livro

router.post('/', (req, res) => {

    Book.create(req.body)

        .then(book => res.json({msg: 'Livro adicionado com
sucesso'}))

        .catch(err => res.status(400).json({ error: 'Não foi
possível adicionar este livro'}));

});

//PUT api/books/:id

//atualizar livro

router.put('/:id', (req, res) => {
```



```
Book.findByIdAndUpdate(req.params.id, req.body)

    .then(book => res.json({msg: 'Atualizado com sucesso'}))

    .catch(err => res.status(400).json({ error: 'Não foi
possível atualizar a base de dados'}));

});

//DELETE api/books/:id

//deletar livro por id

router.delete('/:id', (req, res) => {

    Book.findByIdAndRemove(req.params.id, req.body)

        .then(book => res.json({msg: 'Livro deletado com
sucesso'}))

        .catch(err => res.status(400).json({ error: 'Não existe
este livro'}));

});

module.exports = router;
```

Modelo de banco de dados

Para interagir com nosso banco de dados, precisamos criar um modelo para cada um de nossos recursos. Então, crie uma pasta chamada **models** na raiz e dentro da pasta **models**, crie um arquivo chamado **Book.js** e atualize com isto:

models/Book.js

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




```
const BookSchema = new mongoose.Schema({

  title: {

    type:String,

    required: true

  },

  isbn: {

    type:String,

    required: true

  },

  author: {

    type:String,

    required: true

  },

  description: {

    type:String,

  },

  published_date: {

    type:Date,

  },

  publisher: {

    type:String,

  },

},
```



```
    updated_date: {  
  
      type: Date,  
  
      default: Date.now  
  
    }  
  
  });  
  
module.exports = Book = mongoose.model('book', BookSchema);
```

Agora, atualize `app.js`, o ponto de entrada do back-end com o seguinte código:

`app.js`

```
const express = require('express');  
  
const connectDB = require('./config/db');  
  
const booksRoutes = require('./routes/api/books');  
  
//Express app  
  
const app = express();  
  
//Middleware  
  
app.use(express.json({ extended: false }));  
  
app.get('/', (req, res) => res.send('Hello world!'));  
  
//Conecta o Banco de Dados
```



```
connectDB();

//Routes

app.use('/api/books', booksRoutes);

const port = process.env.PORT || 8082;

app.listen(port, () => console.log(`Server running on port ${port}`));
```

Em seguida, execute o projeto para ver se está tudo bem neste ponto e você pode testar todas as APIs por meio do [Postman](#) ou a extensão **Thunder Client** do VSCode. É importante observar que antes de testar APIs usando o Postman, você precisa primeiro executar o projeto.

<https://expressjs.com/en/api.html#express.urlencoded>

Construindo o Front-End (Client)

Até agora tudo bem! Agora que configuramos nosso back-end, é hora de fazer a transição para a parte do front-end deste tutorial de pilha MERN. Nesta seção, usaremos o React para construir nossas interfaces de usuário. Usaremos Create React App para gerar nossa configuração de arquivo inicial.

Defina qualquer diretório usando um terminal onde você deseja manter todos os arquivos deste projeto e execute `$ npx create-react-app client` para obter o arquivo de configuração inicial.



Agora que estamos no diretório do projeto, podemos usar os comandos disponíveis. Se estiver usando o Yarn, digite `$ yarn start`. Se você estiver usando npm, use `$ npm start`.

Adicionando Bootstrap e Font Awesome ao seu aplicativo React

Temos nosso arquivo de configuração inicial para a parte do frontend. Agora, podemos começar a integrar nosso back-end com nosso front-end. Antes disso, quero adicionar o CDN do [Bootstrap](#) e do [Font Awesome](#) ao nosso projeto.

Abra o arquivo chamado `index.html`, que está na pasta pública `client/public/index.html`, e substitua tudo pelo seguinte código:

`index.html`

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico"
  />

    <meta name="viewport" content="width=device-width,
initial-scale=1" />

    <meta name="theme-color" content="#000000" />

    <!--

      manifest.json provides metadata used when your web app
is installed on a
```



user's mobile device or desktop. See <https://developers.google.com/web/fundamentals/web-app-manifest/>

```
-->
```

```
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
```

```
<!--
```

Notice the use of `%PUBLIC_URL%` in the tags above.

It will be replaced with the URL of the ``public`` folder during the build.

Only files inside the ``public`` folder can be referenced from the HTML.

Unlike `"/favicon.ico"` or `"favicon.ico"`, `"%PUBLIC_URL%/favicon.ico"` will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running ``npm run build``.

```
-->
```

```
<!-- bootstrap css cdn -->
```

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6J
Xm" crossorigin="anonymous">
```



```
<!-- fontawesome cdn -->

<link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
integrity="sha384-
hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54bOGbi
DQ" crossorigin="anonymous">

<title>MERN Bookstore</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this
app.</noscript>

<div id="root"></div>

<!-- bootstrap JS cdn -->

<script src="https://code.jquery.com/jquery-
3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5K
kN" crossorigin="anonymous"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/u
md/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b
4Q" crossorigin="anonymous"></script>

<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootst
rap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCm
Yl" crossorigin="anonymous"></script>
```



```
</body>  
</html>
```

Nosso frontend terá as seguintes funcionalidades:

- Adicionar, criar ou salvar um novo livro
- Mostrar todos os livros que armazenamos no banco de dados
- Mostrar um único livro
- Atualizar um livro
- Excluir um livro

Agora, use o seguinte comando para adicionar algumas dependências necessárias:

```
$ npm install --save react-router-dom
```

```
$ npm install --save axios
```

Por que Axios?

[Axios](#) é um cliente HTTP leve para Node.js e o navegador, semelhante a uma [API Fetch](#). Axios é uma biblioteca async/await baseada em promessa para código assíncrono legível. Podemos integrá-lo facilmente ao React e é fácil de usar em qualquer estrutura de front-end. Chamaremos nossas APIs por meio do Axios.

Existem várias razões pelas quais o Axios é amplamente utilizado. Um dos maiores benefícios do Axios é sua compatibilidade com versões anteriores. Navegadores antigos, como o IE11, também podem executar facilmente o Axios, pois o pacote emite o XMLHttpRequest sob o capô.



O Axios também restringe automaticamente a carga útil ao [enviar uma solicitação](#). Mas, quando você estiver usando Fetch API, é importante que você converta a carga em JSON.

O arquivo Package.json

Neste ponto, nosso arquivo `package.json` deve ser semelhante (mas pode ser diferente) ao código abaixo:

`client - package.json`

```
{  
  
  "name": "mern_a_to_z_client",  
  
  "version": "0.1.0",  
  
  "private": true,  
  
  "dependencies": {  
  
    "@testing-library/jest-dom": "^5.16.5",  
  
    "@testing-library/react": "^13.4.0",  
  
    "@testing-library/user-event": "^13.5.0",  
  
    "axios": "^1.2.1",  
  
    "react": "^18.2.0",  
  
    "react-dom": "^18.2.0",  
  
    "react-router-dom": "^6.4.5",  
  
    "react-scripts": "5.0.1",  
  
    "web-vitals": "^2.1.4"
```




```
},  
  
"scripts": {  
  
  "start": "react-scripts start",  
  
  "build": "react-scripts build",  
  
  "test": "react-scripts test",  
  
  "eject": "react-scripts eject"  
  
},  
  
"eslintConfig": {  
  
  "extends": [  
  
    "react-app",  
  
    "react-app/jest"  
  
  ]  
  
},  
  
"browserslist": {  
  
  "production": [  
  
    ">0.2%",  
  
    "not dead",  
  
    "not op_mini all"  
  
  ],  
  
  "development": [  
  
    "last 1 chrome version",  
  
    "last 1 firefox version",  
  
    "last 1 safari version"
```



```
]

}
```

Criando a pasta components e arquivos

Dentro da pasta `src` (`client/src/`), crie outra pasta chamada `components`, e dentro dela, crie cinco arquivos diferentes:

- `CreateBook.js`
- `ShowBookList.js`
- `BookCard.js`
- `ShowBookDetails.js`
- `UpdateBookInfo.js`

Vamos trabalhar com esses cinco arquivos um pouco mais tarde.

Configurando as rotas

Abra a pasta `App.js` dentro da pasta `src` (`client/src/App.js`) e substitua-a pelo seguinte código:

App.js

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import './App.css';

import CreateBook from './components/CreateBook';

import ShowBookList from './components/ShowBookList';

import ShowBookDetails from './components/ShowBookDetails';
```



```
import UpdateBookInfo from './components/UpdateBookInfo';

const App = () => {

  return (

    <Router>

      <div>

        <Routes>

          <Route exact path="/" element={<ShowBookList />} />

          <Route path="/create-book" element={<CreateBook />} />

          <Route path="/edit-book/:id"
element={<UpdateBookInfo />} />

          <Route path="/show-book/:id"
element={<ShowBookDetails />} />

        </Routes>

      </div>

    </Router>

  );

};

export default App;
```

Aqui, definimos todas as rotas. Para uma definição de caminho específica, seu componente correspondente será renderizado. Ainda não implementamos esses arquivos e componentes — acabamos de concluir a configuração do caminho.



Atualizando o arquivo CSS

Em seguida, atualize um arquivo CSS chamado `App.css` na pasta `src` com o seguinte código:

App.css

```
.App {  
  text-align: center;  
}  
  
.App-logo {  
  animation: App-logo-spin infinite 20s linear;  
  height: 40vmin;  
  pointer-events: none;  
}  
  
.App-header {  
  background-color: #282c34;  
  min-height: 100vh;  
  display: flex;  
  flex-direction: column;
```



```
align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;
}

.App-link {

  color: #61dafb;
}

@keyframes App-logo-spin {

  from {

    transform: rotate(0deg);

  }

  to {

    transform: rotate(360deg);

  }

}

.CreateBook {

  background-color: #2c3e50;

  min-height: 100vh;

  color: white;
```



```
}

.ShowBookDetails {

    background-color: #2c3e50;

    min-height: 100vh;

    color: white;

}

.UpdateBookInfo {

    background-color: #2c3e50;

    min-height: 100vh;

    color: white;

}

.ShowBookList {

    background-color: #2c3e50;

    height: 100%;

    width: 100%;

    min-height: 100vh;

    min-width: 100px;

    color: white;

}
```



```
/* BookList Styles */

.list {

    display: grid;

    margin: 20px 0 50px 0;

    grid-template-columns: repeat(4, 1fr);

    grid-auto-rows: 1fr;

    grid-gap: 2em;

}

.card-container {

    width: 250px;

    border: 1px solid rgba(0,0,.125);

    margin: 0 auto;

    border-radius: 5px;

    overflow: hidden;

}

.desc {

    height: 130px;

    padding: 10px;

}

.desc h2 {
```



```
font-size: 1em;

font-weight: 400;
}

.desc h3, p {

  font-weight: 300;
}

.desc h3 {

  color: #6c757d;

  font-size: 1em;

  padding: 10px 0 10px 0;
}
```

Adicionando nossos componentes de recursos

Agora, é hora de adicionar componentes de recursos ao nosso projeto de pilha MERN. Nosso arquivo `CreateBook.js` é responsável por adicionar, criar ou salvar um novo livro ou as informações de um livro.

`CreateBook.js`

Portanto, atualize `CreateBook.js` com o seguinte código:

```
import React, { useState } from 'react';

import { Link } from 'react-router-dom';
```




```
import axios from 'axios';

import { useNavigate } from 'react-router-dom';

const CreateBook = (props) => {

  // Define the state with useState hook

  const navigate = useNavigate();

  const [book, setBook] = useState({

    title: '',

    isbn: '',

    author: '',

    description: '',

    published_date: '',

    publisher: '',

  });

  const onChange = (e) => {

    setBook({ ...book, [e.target.name]: e.target.value });

  };

  const onSubmit = (e) => {

    e.preventDefault();
```



```
    axios

      .post('http://localhost:8082/api/books', book)

      .then((res) => {

        setBook({

          title: '',

          isbn: '',

          author: '',

          description: '',

          published_date: '',

          publisher: '',

        });

        // Push to /

        navigate('/');

      })

      .catch((err) => {

        console.log('Error in CreateBook!');

      });

    };

    return (

      <div className='CreateBook'>

        <div className='container'>
```



```
<div className='row'>

  <div className='col-md-8 m-auto'>

    <br />

    <Link to='/' className='btn btn-outline-warning
float-left'>

      Show Book List

    </Link>

  </div>

  <div className='col-md-8 m-auto'>

    <h1 className='display-4 text-center'>Add
Book</h1>

    <p className='lead text-center'>Create new
book</p>

    <form noValidate onSubmit={onSubmit}>

      <div className='form-group'>

        <input

          type='text'

          placeholder='Title of the Book'

          name='title'

          className='form-control'

          value={book.title}

          onChange={onChange}

        />

      </div>

    </form>

  </div>

</div>
```

```
</div>
```

```
<br />
```

```
<div className='form-group'>
```

```
<input
```

```
  type='text'
```

```
  placeholder='ISBN'
```

```
  name='isbn'
```

```
  className='form-control'
```

```
  value={book.isbn}
```

```
  onChange={onChange}
```

```
/>
```

```
</div>
```

```
<div className='form-group'>
```

```
<input
```

```
  type='text'
```

```
  placeholder='Author'
```

```
  name='author'
```

```
  className='form-control'
```

```
  value={book.author}
```

```
  onChange={onChange}
```

```
/>
```

```
</div>
```

```
<div className='form-group'>
```

```
<input
```

```
  type='text'
```

```
  placeholder='Describe this book'
```

```
  name='description'
```

```
  className='form-control'
```

```
  value={book.description}
```

```
  onChange={onChange}
```

```
</div>
```

```
<div className='form-group'>
```

```
<input
```

```
  type='date'
```

```
  placeholder='published_date'
```

```
  name='published_date'
```

```
  className='form-control'
```

```
  value={book.published_date}
```

```
  onChange={onChange}
```

```
</div>
```

```
<div className='form-group'>

  <input

    type='text'

    placeholder='Publisher of this Book'

    name='publisher'

    className='form-control'

    value={book.publisher}

    onChange={onChange}

  />

</div>

<input

  type='submit'

  className='btn btn-outline-warning btn-block
mt-4'

  />

</form>

</div>

</div>

</div>

);

};
```



```
export default CreateBook;
```

ShowBookList.js

O componente `ShowBookList.js` será responsável por mostrar todos os livros que já temos armazenados em nosso banco de dados. Atualize `ShowBookList.js` com este código:

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

import '../App.css';

import axios from 'axios';

import { Link } from 'react-router-dom';

import BookCard from './BookCard';

function ShowBookList() {

  const [books, setBooks] = useState([]);

  useEffect(() => {

    axios

      .get('http://localhost:8082/api/books')

      .then((res) => {

        setBooks(res.data);

      })

      .catch((err) => {
```



```
        console.log('Error from ShowBookList');

    });

}, []);

const bookList =

    books.length === 0

    ? 'there is no book record!'

    : books.map((book, k) => <BookCard book={book} key={k}
/>);

return (

    <div className='ShowBookList'>

        <div className='container'>

            <div className='row'>

                <div className='col-md-12'>

                    <br />

                    <h2 className='display-4 text-center'>Books
List</h2>

                </div>

                <div className='col-md-11'>

                    <Link

                        to='/create-book'
```




```
        className='btn btn-outline-warning float-right'

        >

        + Add New Book

    </Link>

    <br />

    <br />

    <hr />

</div>

</div>

<div className='list'>{bookList}</div>

</div>

</div>

);
}

export default ShowBookList;
```

BookCard.js

Aqui, usamos um componente funcional chamado `BookCard.js`, que pega as informações de um livro de `ShowBookList.js` e cria um cartão para cada livro. Escreva o seguinte código para atualizar seu arquivo `BookCard.js`:

```
import React from 'react';
```



```
import { Link } from 'react-router-dom';

import '../App.css';

const BookCard = (props) => {

  const book = props.book;

  return (

    <div className='card-container'>

      <img

        src='https://images.unsplash.com/photo-1495446815901-a7297e633e8d'

        alt='Books'

        height={200}

      />

      <div className='desc'>

        <h2>

          <Link to={`/${show-
book/${book._id}`}>{book.title}</Link>

        </h2>

        <h3>{book.author}</h3>

        <p>{book.description}</p>

      </div>

    </div>

  )
}
```



```
);  
  
};  
  
export default BookCard;
```

NOTA: Aqui, usei o mesmo img src para cada livro, pois a respectiva imagem de cada livro pode nem sempre estar disponível. Mude a fonte da imagem e você também pode usar uma imagem diferente para cada livro.

ShowBookDetails

O componente `ShowBookDetails` tem uma tarefa: mostrar todas as informações que temos sobre qualquer livro. Temos os botões `delete` e `edit` aqui para obter acesso:

```
import React, { useState, useEffect } from 'react';  
  
import { Link, useParams, useNavigate } from 'react-router-dom';  
  
import '../App.css';  
  
import axios from 'axios';  
  
function ShowBookDetails(props) {  
  
  const [book, setBook] = useState({});  
  
  const { id } = useParams();  
  
  const navigate = useNavigate();
```



```
useEffect(() => {

  axios

    .get(`http://localhost:8082/api/books/${id}`)

    .then((res) => {

      setBook(res.data);

    })

    .catch((err) => {

      console.log('Error from ShowBookDetails');

    });

}, [id]);

const onDeleteClick = (id) => {

  axios

    .delete(`http://localhost:8082/api/books/${id}`)

    .then((res) => {

      navigate('/');

    })

    .catch((err) => {

      console.log('Error form ShowBookDetails_deleteClick');

    });

};
```



```
const BookItem = (  
  
  <div>  
  
    <table className='table table-hover table-dark'>  
  
      <tbody>  
  
        <tr>  
  
          <th scope='row'>1</th>  
  
          <td>Title</td>  
  
          <td>{book.title}</td>  
  
        </tr>  
  
        <tr>  
  
          <th scope='row'>2</th>  
  
          <td>Author</td>  
  
          <td>{book.author}</td>  
  
        </tr>  
  
        <tr>  
  
          <th scope='row'>3</th>  
  
          <td>ISBN</td>  
  
          <td>{book.isbn}</td>  
  
        </tr>  
  
        <tr>  
  
          <th scope='row'>4</th>  
  
          <td>Publisher</td>  
  
          <td>{book.publisher}</td>  

```

```
</tr>

<tr>

  <th scope='row'>5</th>

  <td>Published Date</td>

  <td>{book.published_date}</td>

</tr>

<tr>

  <th scope='row'>6</th>

  <td>Description</td>

  <td>{book.description}</td>

</tr>

</tbody>

</table>

</div>

);

return (

  <div className='ShowBookDetails'>

    <div className='container'>

      <div className='row'>

        <div className='col-md-10 m-auto'>

          <br /> <br />

        </div>

      </div>

    </div>

  </div>

);
```



```
        <Link to="/" className='btn btn-outline-warning
float-left'>

        Show Book List

    </Link>

</div>

<br />

<div className='col-md-8 m-auto'>

    <h1 className='display-4 text-center'>Book's
Record</h1>

    <p className='lead text-center'>View Book's
Info</p>

    <hr /> <br />

</div>

<div className='col-md-10 m-auto'>{BookItem}</div>

<div className='col-md-6 m-auto'>

    <button

        type='button'

        className='btn btn-outline-danger btn-lg btn-
block'

        onClick={() => {

            onDeleteClick(book._id);

        }}

    >

    Delete Book
```

```
        </button>

    </div>

    <div className='col-md-6 m-auto'>

        <Link

            to={`/edit-book/${book._id}`}

            className='btn btn-outline-info btn-lg btn-
block'

        >

            Edit Book

        </Link>

    </div>

</div>

</div>

</div>

);
}

export default ShowBookDetails;
```

UpdateBookInfo.js

UpdateBookInfo.js, como o próprio nome indica, é responsável por atualizar as informações de um livro. Um botão Editar livro acionará a execução desse



componente. Depois de clicar em Editar livro, veremos um formulário com as informações antigas, que poderemos editar ou substituir:

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

import { Link, useParams, useNavigate } from "react-router-dom";

import axios from "axios";

import "../App.css";

function UpdateBookInfo(props) {

  const [book, setBook] = useState({

    title: "",

    isbn: "",

    author: "",

    description: "",

    published_date: "",

    publisher: "",

  });

  const { id } = useParams();

  const navigate = useNavigate();

  useEffect(() => {

    axios
```



```
.get(`http://localhost:8082/api/books/${id}`)

.then((res) => {

  setBook({

    title: res.data.title,

    isbn: res.data.isbn,

    author: res.data.author,

    description: res.data.description,

    published_date: res.data.published_date,

    publisher: res.data.publisher,

  });

})

.catch((err) => {

  console.log("Error from UpdateBookInfo");

});

}, [id]);

const onChange = (e) => {

  setBook({ ...book, [e.target.name]: e.target.value });

};

const onSubmit = (e) => {

  e.preventDefault();
```



```
const data = {

  title: book.title,

  isbn: book.isbn,

  author: book.author,

  description: book.description,

  published_date: book.published_date,

  publisher: book.publisher,

};

axios

  .put(`http://localhost:8082/api/books/${id}`, data)

  .then((res) => {

    navigate(`/show-book/${id}`);

  })

  .catch((err) => {

    console.log("Error in UpdateBookInfo!");

  });

};

return (

  <div className="UpdateBookInfo">

    <div className="container">

      <div className="row">
```



```
<div className="col-md-8 m-auto">

    <br />

    <Link to="/" className="btn btn-outline-warning
float-left">

        Show Book List

    </Link>

</div>

<div className="col-md-8 m-auto">

    <h1 className="display-4 text-center">Edit
Book</h1>

    <p className="lead text-center">Update Book's
Info</p>

</div>

</div>

<div className="col-md-8 m-auto">

    <form noValidate onSubmit={onSubmit}>

        <div className="form-group">

            <label htmlFor="title">Title</label>

            <input

                type="text"

                placeholder="Title of the Book"

                name="title"

                className="form-control">
```



```
        value={book.title}

        onChange={onChange}

    />

</div>

<br />

<div className="form-group">

    <label htmlFor="isbn">ISBN</label>

    <input

        type="text"

        placeholder="ISBN"

        name="isbn"

        className="form-control"

        value={book.isbn}

        onChange={onChange}

    />

</div>

<br />

<div className="form-group">

    <label htmlFor="author">Author</label>

    <input

        type="text"
```



```
        placeholder="Author"

        name="author"

        className="form-control"

        value={book.author}

        onChange={onChange}

    />

</div>

<br />

<div className="form-group">

    <label htmlFor="description">Description</label>

    <textarea

        type="text"

        placeholder="Description of the Book"

        name="description"

        className="form-control"

        value={book.description}

        onChange={onChange}

    />

</div>

<br />

<div className="form-group">
```



```
<label htmlFor="published_date">Published
Date</label>

<input

  type="text"

  placeholder="Published Date"

  name="published_date"

  className="form-control"

  value={book.published_date}

  onChange={onChange}

/>

</div>

<br />

<div className="form-group">

  <label htmlFor="publisher">Publisher</label>

  <input

    type="text"

    placeholder="Publisher of the Book"

    name="publisher"

    className="form-control"

    value={book.publisher}

    onChange={onChange}

  />
```



```
        </div>

        <br />

        <button

            type="submit"

            className="btn btn-outline-info btn-lg btn-
block"

        >

            Update Book

        </button>

    </form>

</div>

</div>

</div>

);
}

export default UpdateBookInfo;
```

Conectando e executando o front-end para o back-end

Acabamos de implementar todos os nossos componentes! Agora, precisamos de uma pequena mudança em nosso projeto de back-end do lado do servidor.

Se tentarmos chamar nossa API de back-end da parte de front-end, ocorrerá um erro:



```
"Access to XMLHttpRequest at 'http://localhost:8082/api/books'
from origin 'http://localhost:3000' has been blocked by CORS
policy: Response to preflight request doesn't pass access
control check: No 'Access-Control-Allow-Origin' header is
present on the requested resource."
```

Para resolver isso, precisamos instalar `cors` em nosso projeto do lado do servidor de back-end. Vá para a pasta do projeto e execute `$ npm install cors`.

Agora, atualize `app.js`, o ponto de entrada do back-end com o seguinte código:

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

const connectDB = require('./config/db');

const booksRoutes = require('./routes/api/books');

const cors = require('cors');

//Express app

const app = express();

//Middleware

app.use(express.json({ extended: false }));

app.get('/', (req, res) => res.send('Hello world!'));

//Conecta o Banco de Dados

connectDB();
```



```
// Cors

app.use(cors({ origin: true, credentials: true }));

//Routes

app.use('/api/books', booksRoutes);

const port = process.env.PORT || 8082;

app.listen(port, () => console.log(`Server running on port ${port}`));
```

Também é importante que você adicione esta linha, `app.use(express.json({ extended: false }));`. O método `express.json` permite que o Express leia os dados enviados usando uma solicitação `POST` ou `PUT`. Ele é usado para reconhecer objetos de entrada como objetos JSON.

Conectando e executando o front-end para o back-end

Siga as etapas abaixo para executar o front-end e o back-end de nosso exemplo de pilha MERN.

Primeiro, execute o servidor (dentro da pasta do projeto):



Se você receber algum erro, siga os comandos abaixo (dentro da pasta do projeto):

```
$ npm install
```

```
$ npm run app
```

Para executar o cliente, execute o comando abaixo no diretório do projeto frontend:

```
$ npm start
```

Se você receber um erro novamente, siga os mesmos comandos abaixo:

```
$ npm install
```

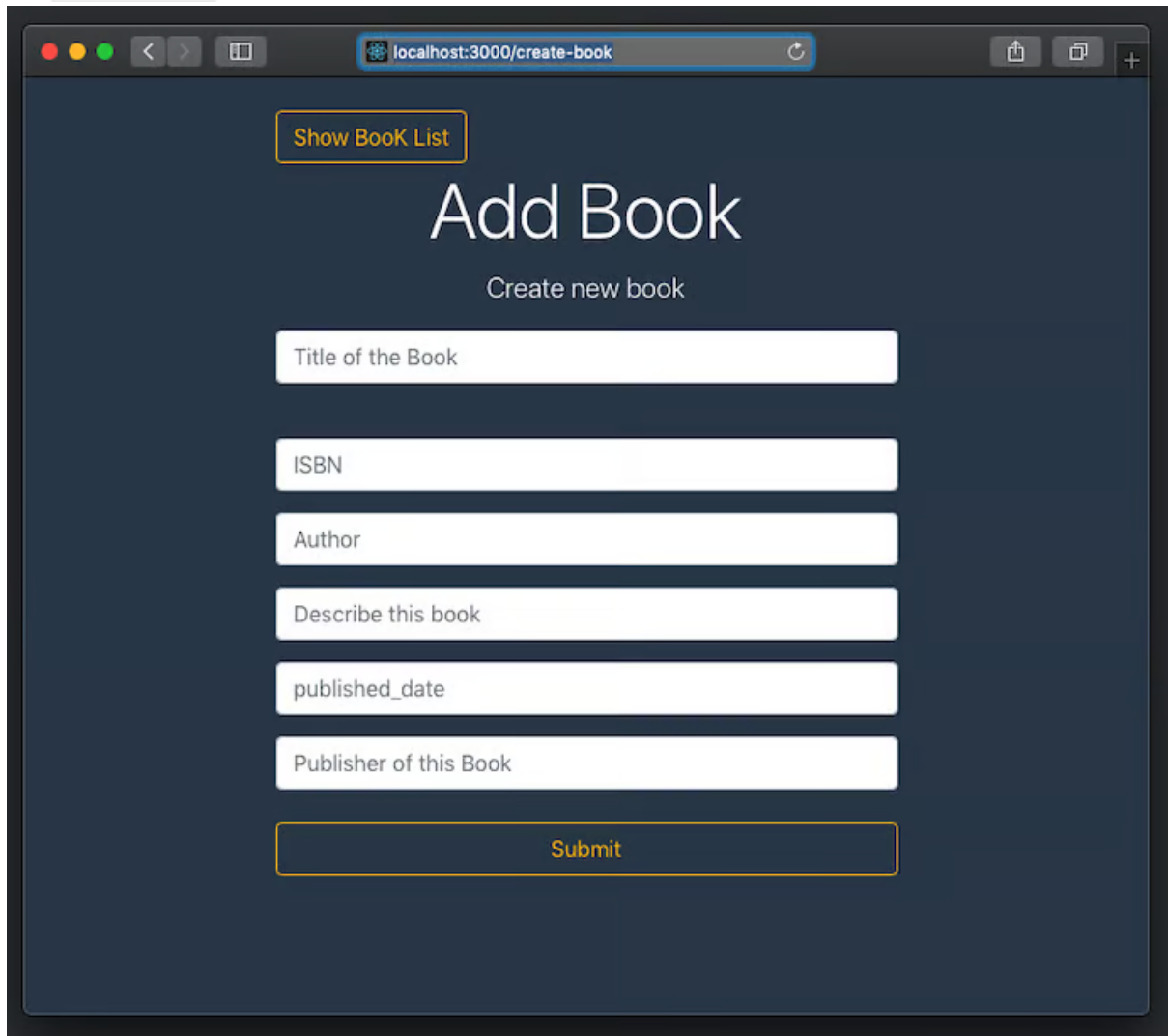
```
$ npm start
```

Testando nosso aplicativo de stack MERN no navegador

Vamos verificar tudo no navegador. Abra <http://localhost:3000> em seu navegador. Agora, você pode adicionar um livro, excluir um livro, mostrar a lista de livros e editar livros. As seguintes rotas devem funcionar de acordo:

Adicionar um livro novo:

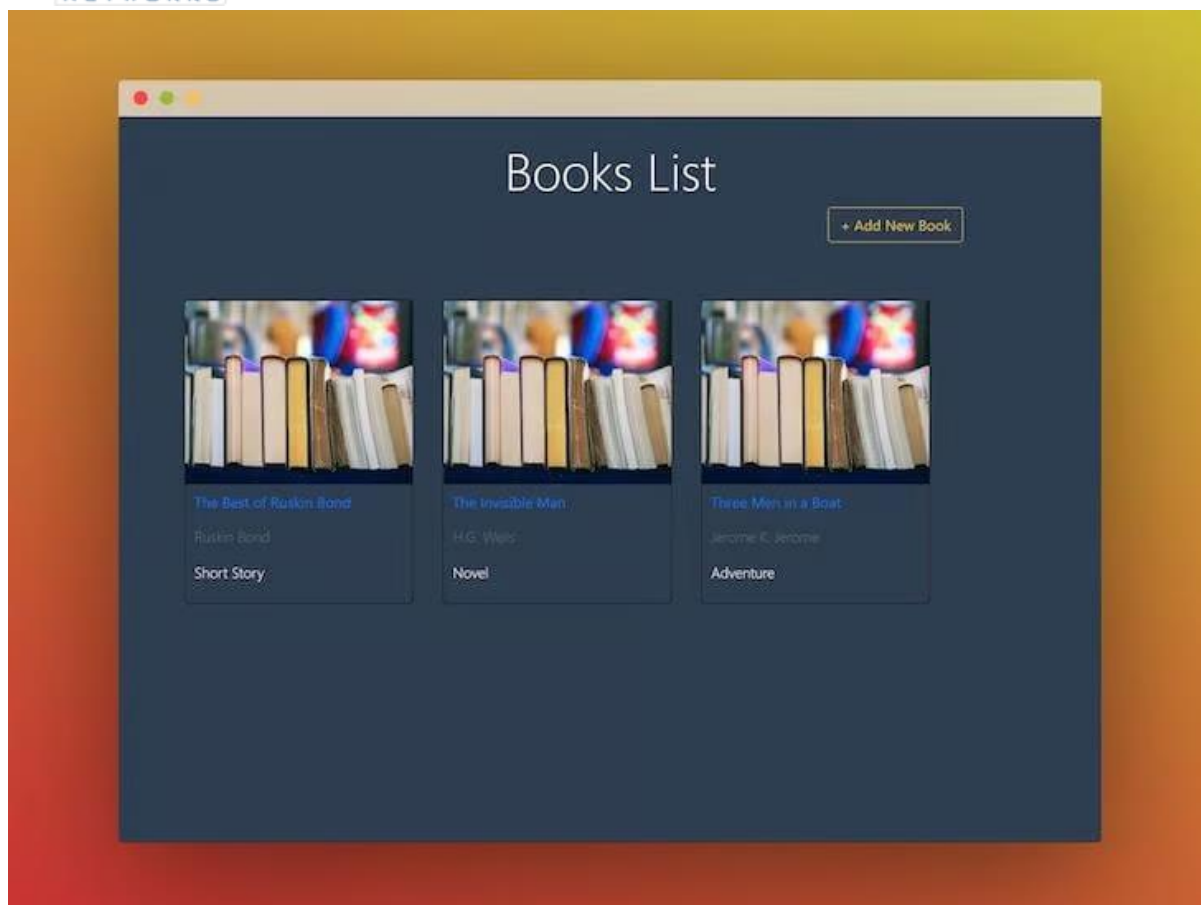
<http://localhost:3000/create-book>



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/create-book'. The page has a dark blue background. At the top left, there is a button labeled 'Show Book List'. The main heading is 'Add Book' in large white text, followed by the subtitle 'Create new book' in smaller white text. Below this, there are six white input fields stacked vertically, each with a label: 'Title of the Book', 'ISBN', 'Author', 'Describe this book', 'published_date', and 'Publisher of this Book'. At the bottom of the form is a wide button labeled 'Submit'.

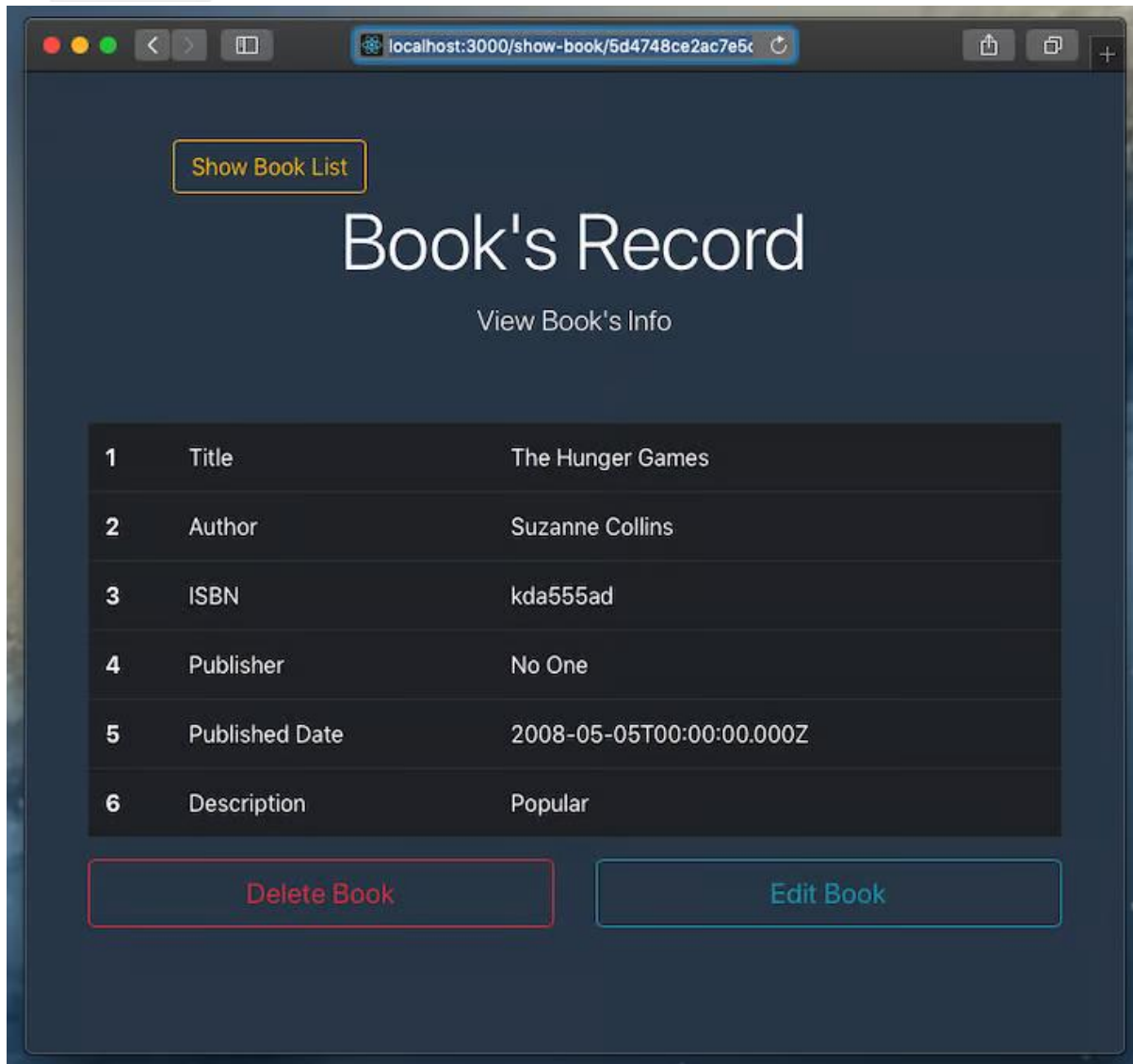
Mostrar a lista de livros:

<http://localhost:3000/>



Mostrar a informação de qualquer livro:

<http://localhost:3000/show-book/:id>

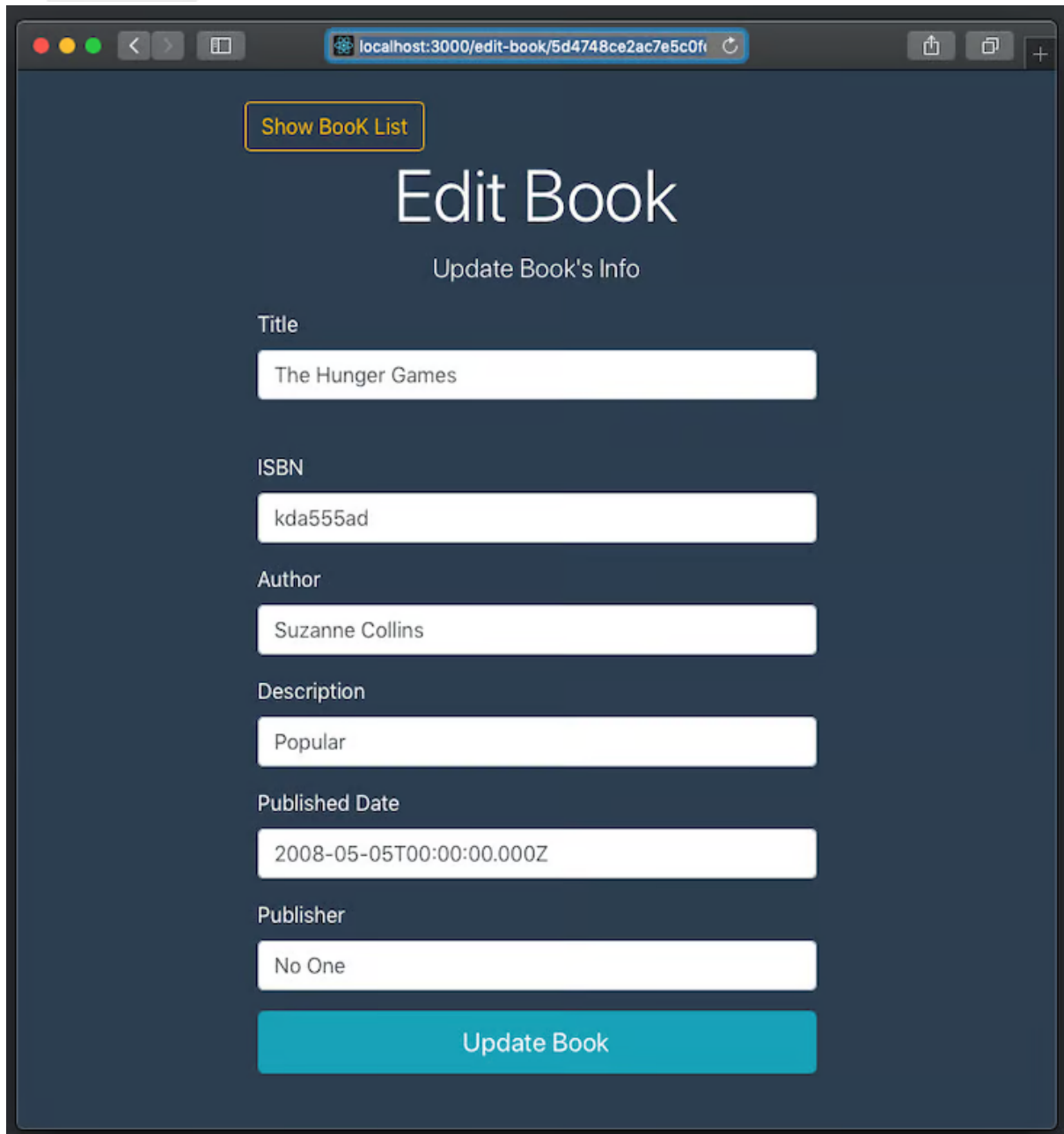


The screenshot shows a web browser window with the address bar displaying `localhost:3000/show-book/5d4748ce2ac7e5c`. The page has a dark blue background. At the top left, there is a button labeled "Show Book List". The main heading is "Book's Record" in a large white font, with the subtitle "View Book's Info" below it. A table with 6 rows displays book details. At the bottom, there are two buttons: "Delete Book" (highlighted with a red border) and "Edit Book" (highlighted with a blue border).

1	Title	The Hunger Games
2	Author	Suzanne Collins
3	ISBN	kda555ad
4	Publisher	No One
5	Published Date	2008-05-05T00:00:00.000Z
6	Description	Popular

Atualizar uma informação:

<http://localhost:3000/edit-book/:id>

A screenshot of a web browser window displaying a form titled "Edit Book". The browser's address bar shows "localhost:3000/edit-book/5d4748ce2ac7e5c0f". The form has a dark blue background and white text. At the top left, there is a button labeled "Show Book List". The main title "Edit Book" is centered, followed by the subtitle "Update Book's Info". The form contains several input fields: "Title" with the value "The Hunger Games", "ISBN" with "kda555ad", "Author" with "Suzanne Collins", "Description" with "Popular", "Published Date" with "2008-05-05T00:00:00.000Z", and "Publisher" with "No One". At the bottom, there is a large teal button labeled "Update Book".

localhost:3000/edit-book/5d4748ce2ac7e5c0f

Show Book List

Edit Book

Update Book's Info

Title

The Hunger Games

ISBN

kda555ad

Author

Suzanne Collins

Description

Popular

Published Date

2008-05-05T00:00:00.000Z

Publisher

No One

Update Book

Parabéns! Você concluiu com sucesso o projeto Full Stack MERN de Livraria.

Este projeto é simples mas tem excelente estrutura para você usá-lo como padrão para fazer outros projetos mais complexos conforme a necessidade do mercado.