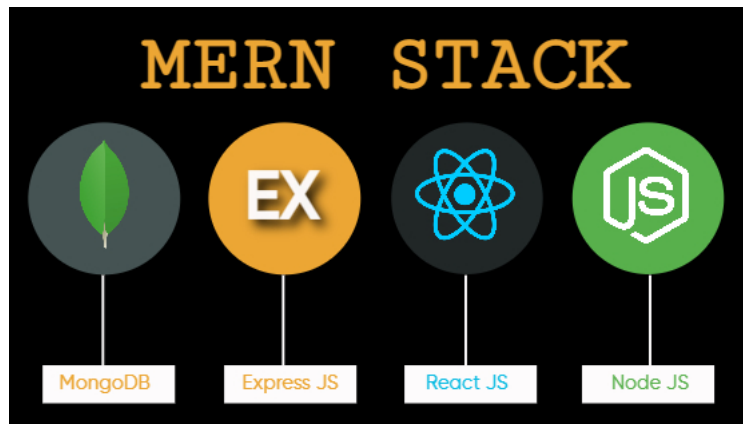


Aula 1 – Mern Stack Exemplo

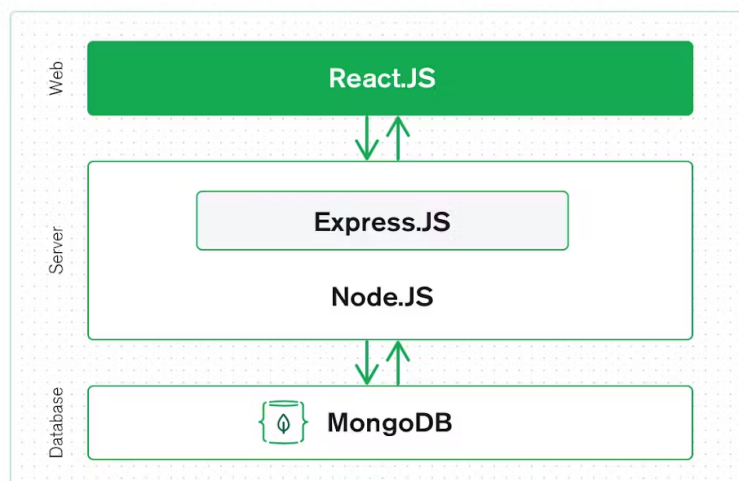
Objetivo da aula:

- Criar uma aplicação FullStack (frontend e backend)
- Conectar com MongoDB
- Deploy de uma aplicação FullStack

Hoje vamos criar uma aplicação MERN que é a abreviação das tecnologias MongoDB, Express JS, React JS e Node JS. Como é uma stack JavaScript que permite o desenvolvimento de aplicações web full-stack utilizando a linguagem JavaScript muito popular, muitas empresas pedem esse tipo de conhecimento para integrar backend e frontend



Em nossa aplicação, vamos adotar a seguinte arquitetura onde teremos uma instancia do MongoDB na nuvem (Atlas) sendo acessada pelo nosso servidor backend permitindo coletar informações do usuário por meio do frontend.

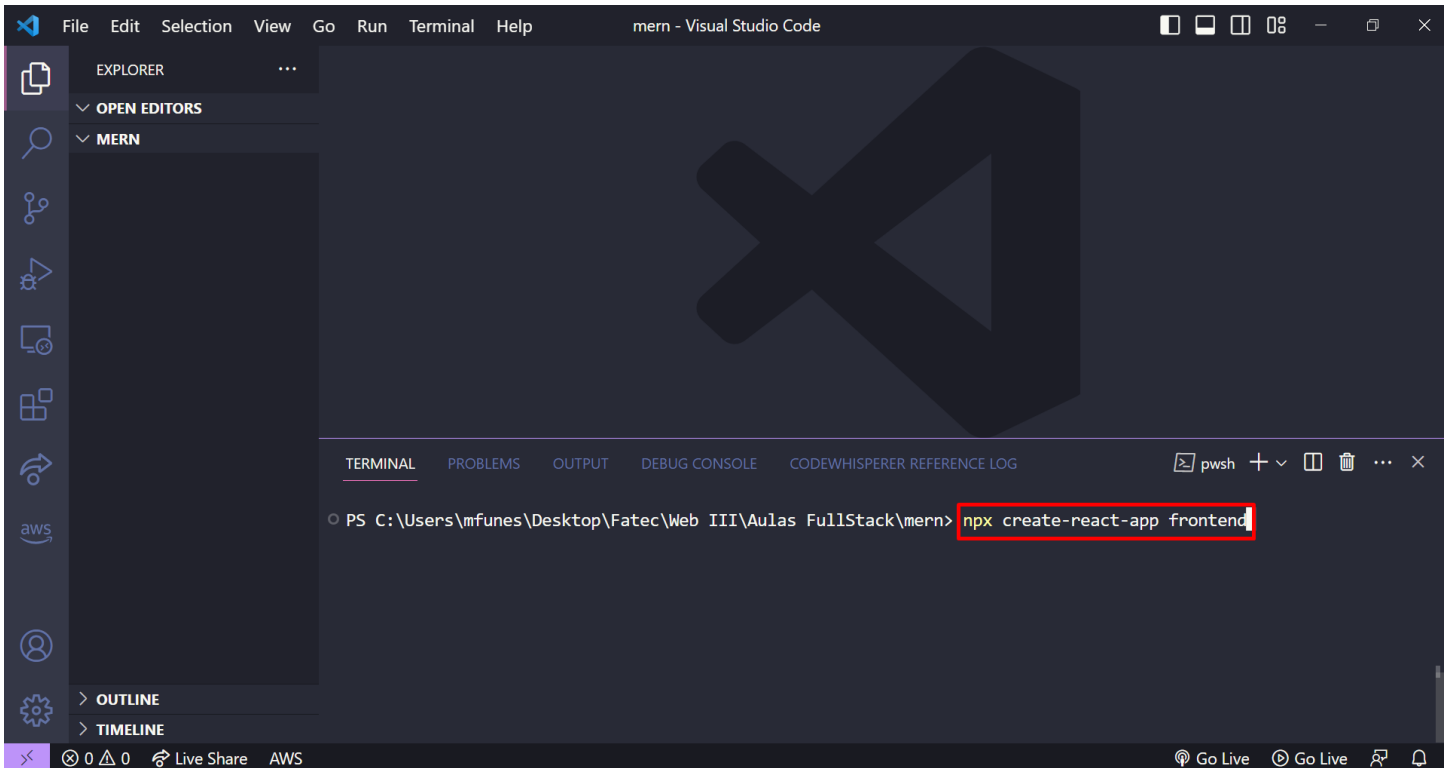


Esse tutorial é o oficial do MongoDB:

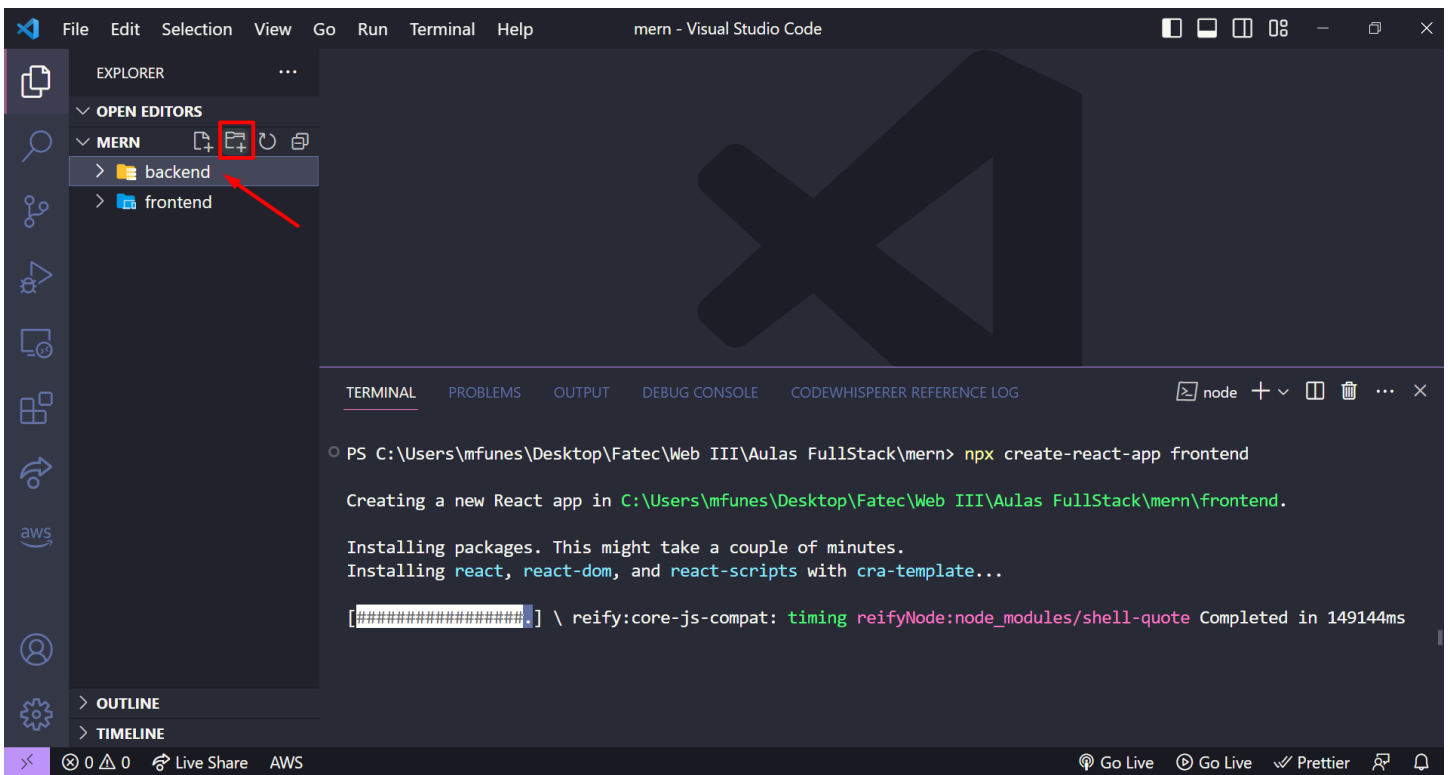
<https://www.mongodb.com/languages/mern-stack-tutorial>

<https://github.com/mongodb-developer/mern-stack-example>

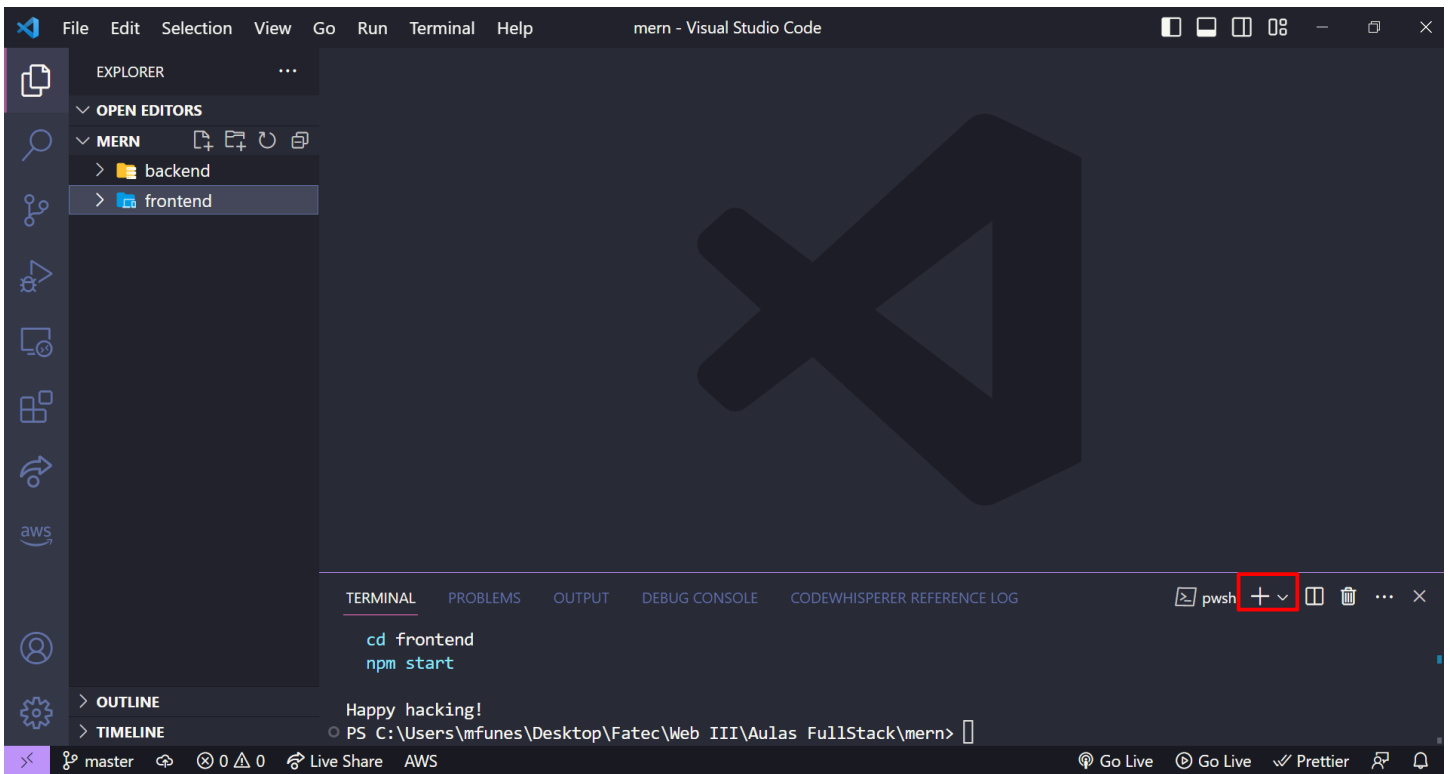
01 – Crie uma pasta chamada mern. Vamos começar subindo nosso frontend, digite no terminal “npx create-react-app frontend” e aguarde a criação do front.



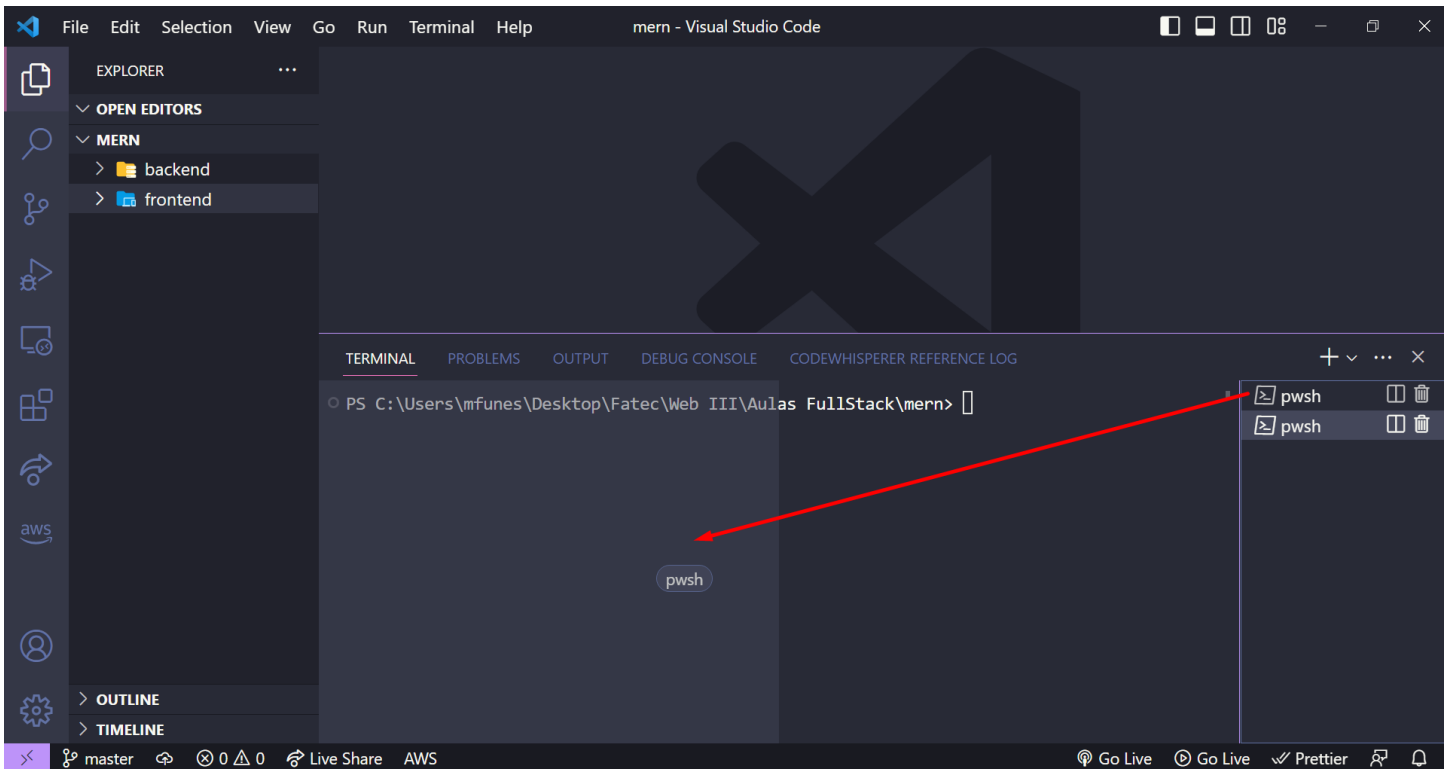
02 – Enquanto aguardamos o frontend, crie uma nova pasta chamada backend dentro da pasta mern.



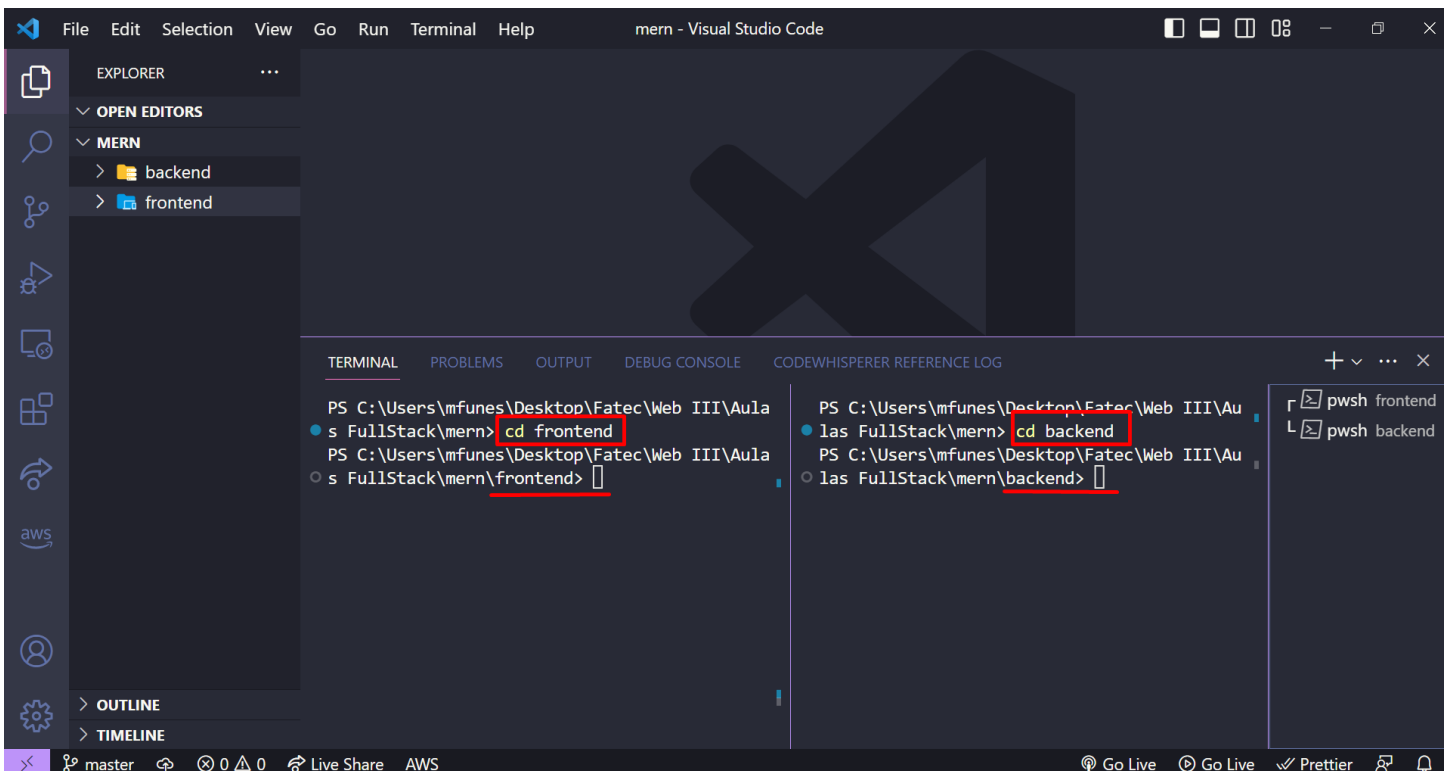
03 – Após o frontend finalizar o setup iniciar, clique no botão de + no terminal, vamos trabalhar com 2 terminais, um para cada pasta.



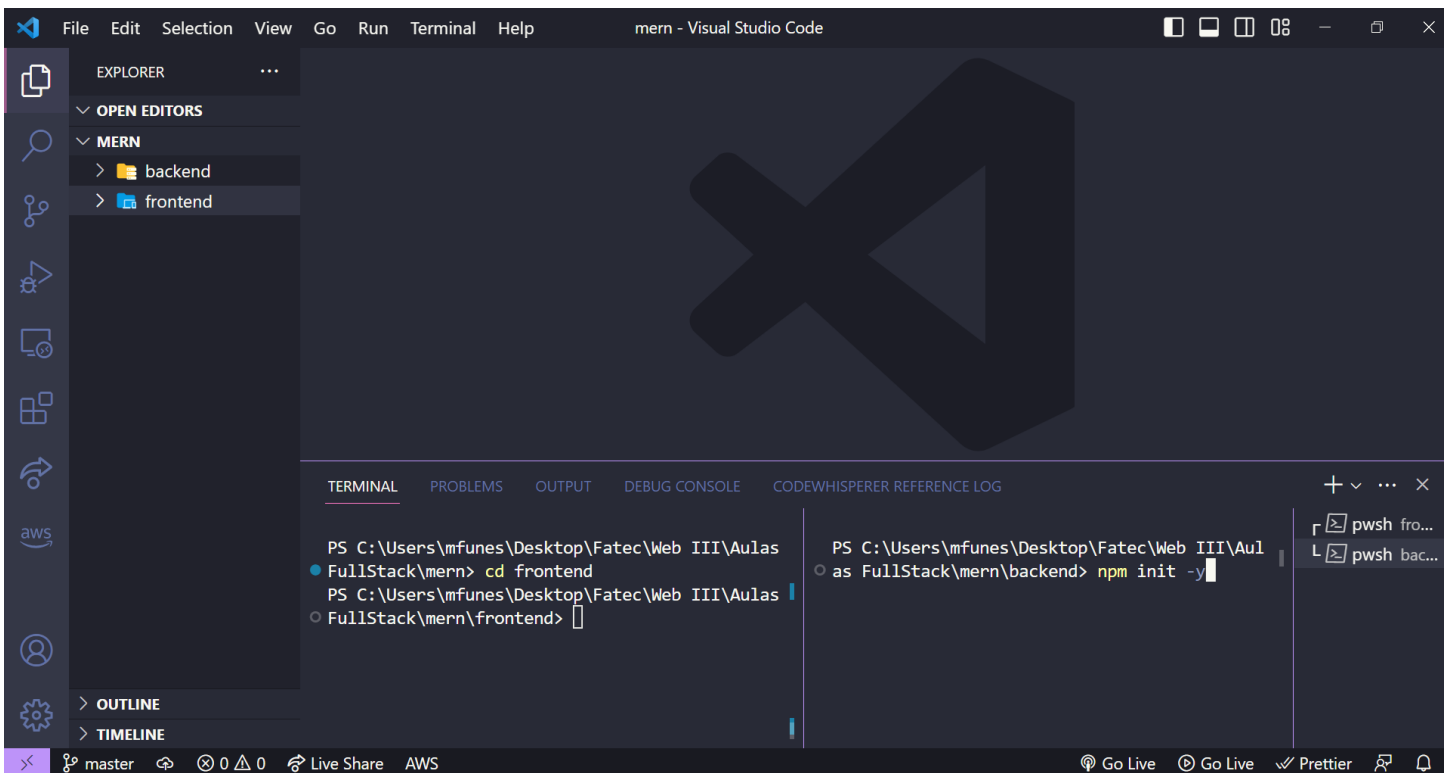
04 – Arraste um dos terminais para a lateral para que ele consiga exibir os 2 terminais lado a lado.



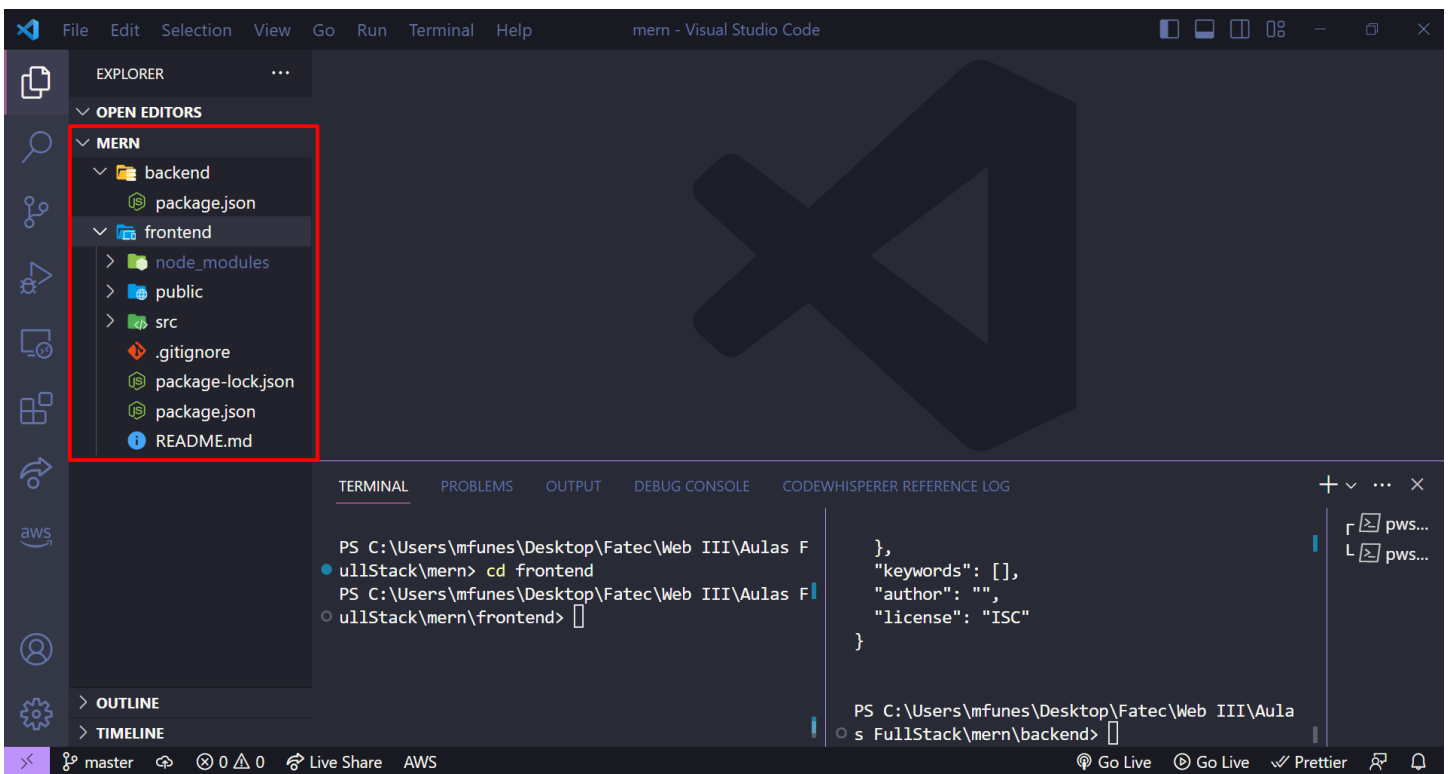
05 – Na primeira janela do terminal digite “cd frontend” e na outra cd backend, assim cada terminal ficará em uma pasta diferente, permitindo o controle de ambos ao mesmo tempo.



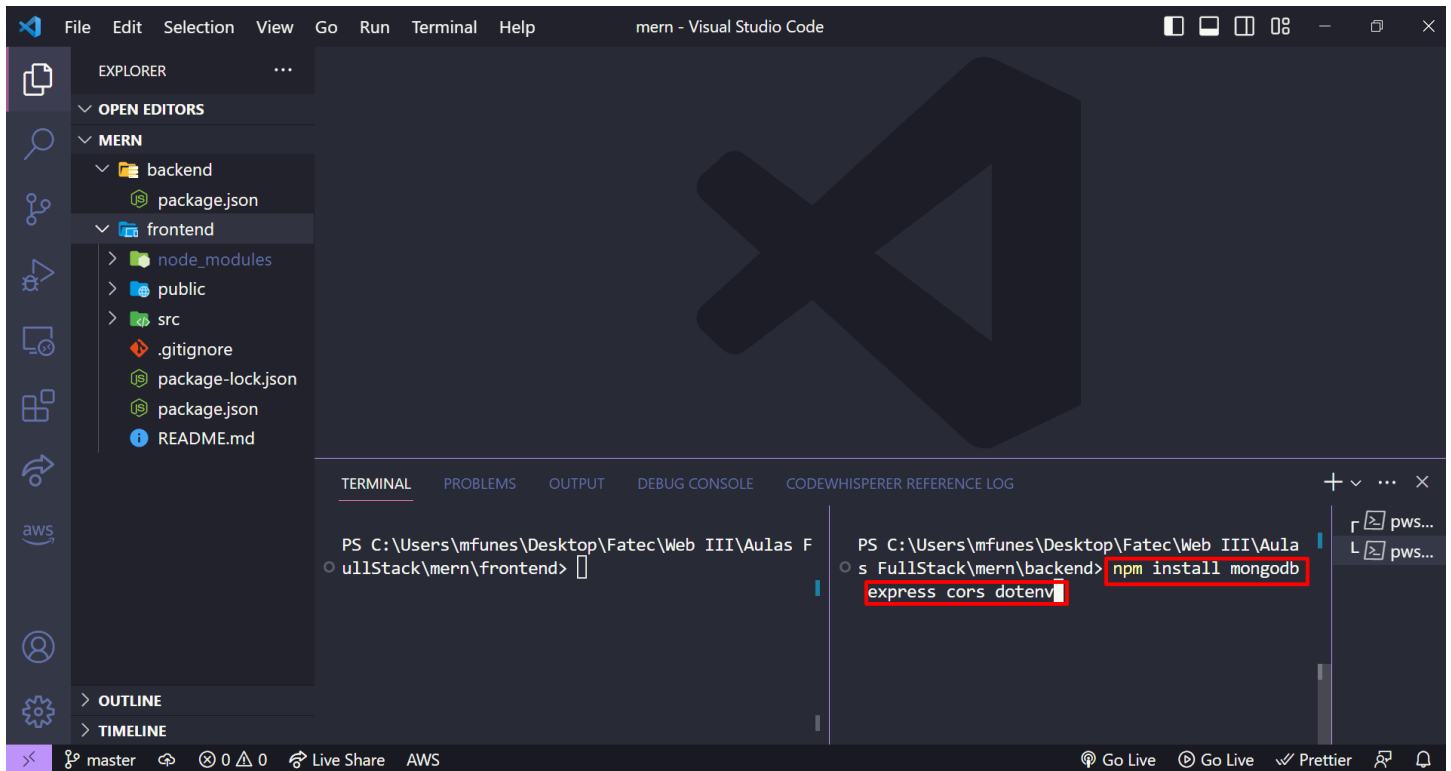
06 – No terminal do backend, dê o comando “npm init -y” para iniciar um servidor node.



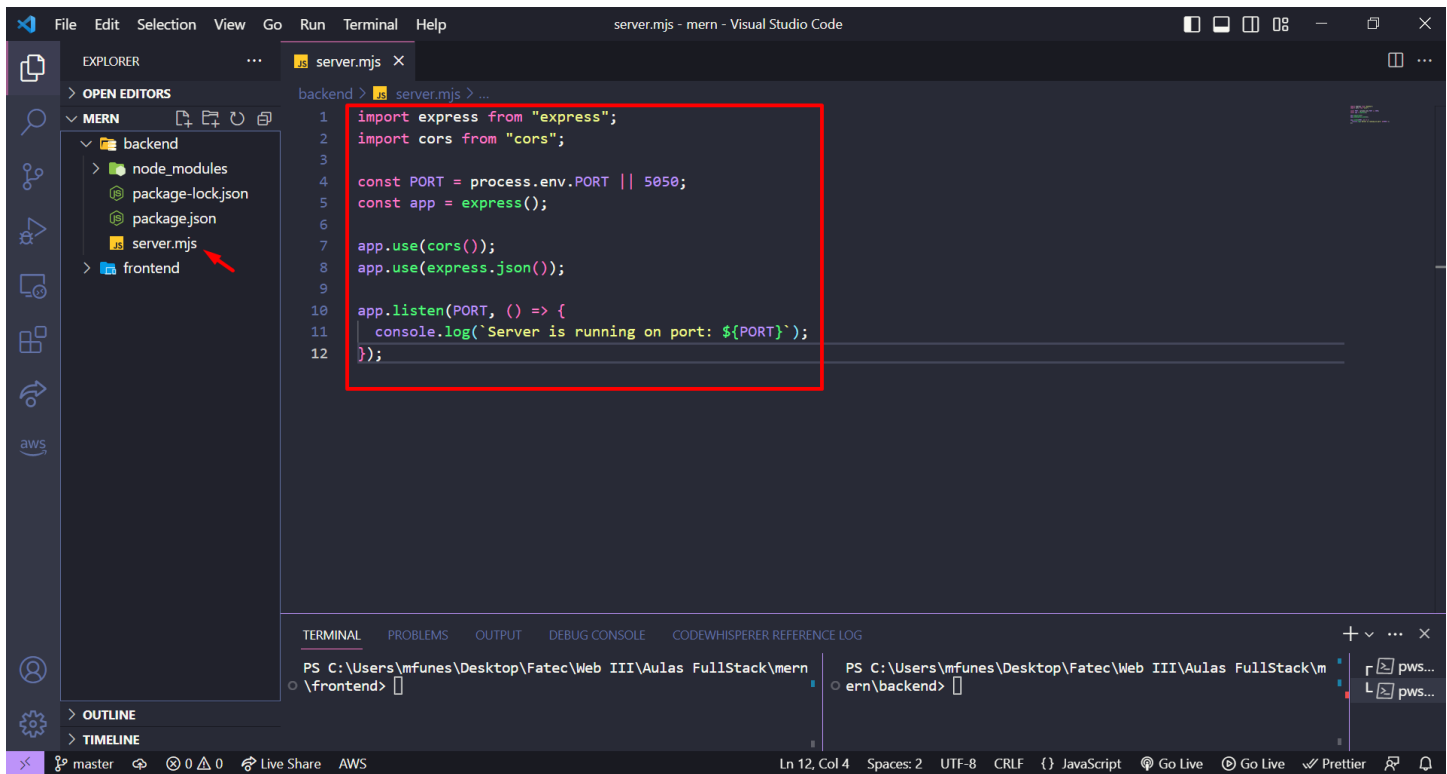
07 – Temos agora o setup backend e frontend da nossa aplicação.



08 – Vamos agora instalar as dependência do Node para termos de fato uma stack MERN em nosso projeto, no terminal do backend digite “npm install mongodb express cors dotenv”

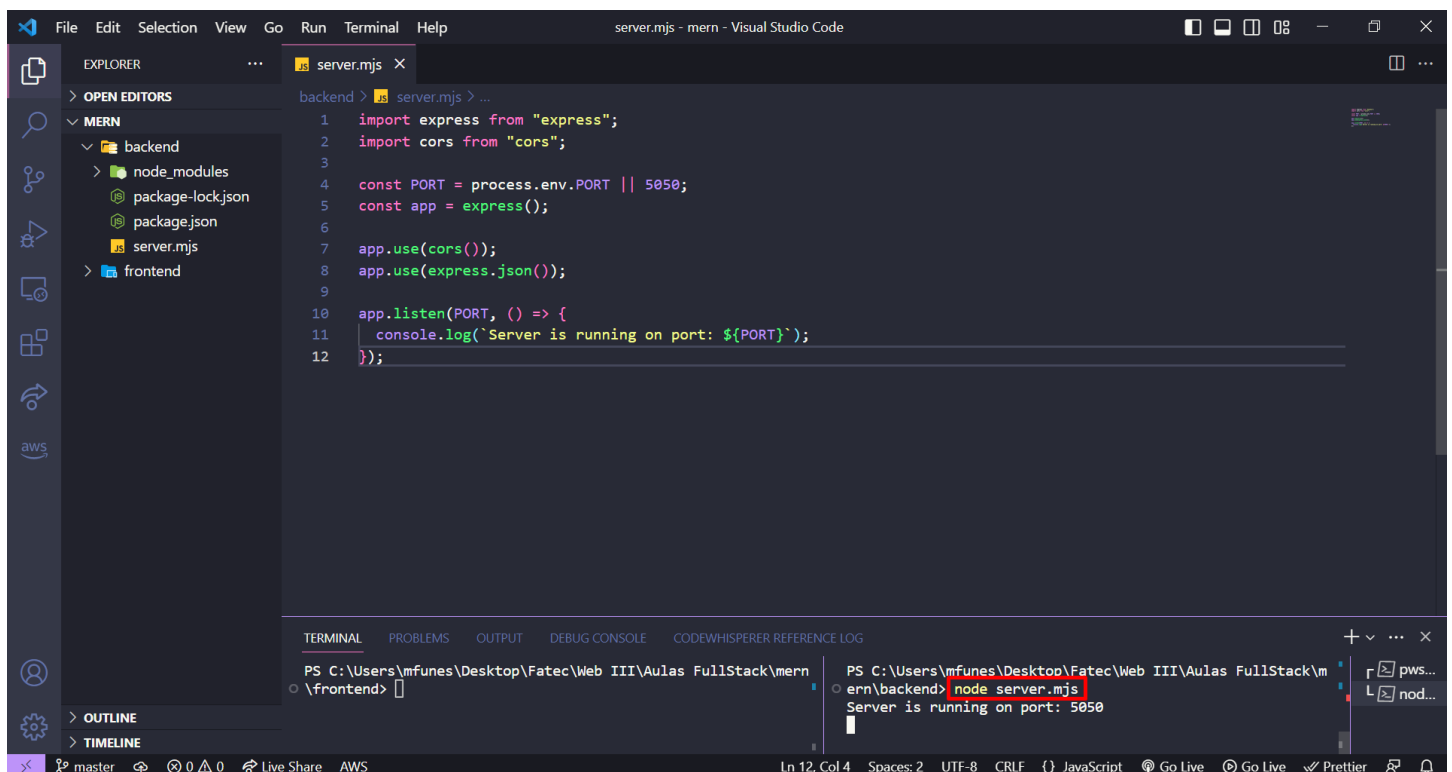


09 – Vamos agora configurar nosso servidor, crie um arquivo chamado server.mjs e faça as importações do express e cors. Além disso vamos definir a porta do servidor. MJS utiliza uma abordagem mais moderna do node chamada ES Modules, veja mais informações aqui: <https://dev.to/oieduardorabelo/usando-es-modules-esm-em-node-js-um-guia-pratico-part-1-3bjp>



```
1 import express from "express";
2 import cors from "cors";
3
4 const PORT = process.env.PORT || 5050;
5 const app = express();
6
7 app.use(cors());
8 app.use(express.json());
9
10 app.listen(PORT, () => {
11   console.log(`Server is running on port: ${PORT}`);
12 });
```

10 – Dê um comando `node server.mjs` no terminal backend e veja se o servidor está funcionando corretamente.

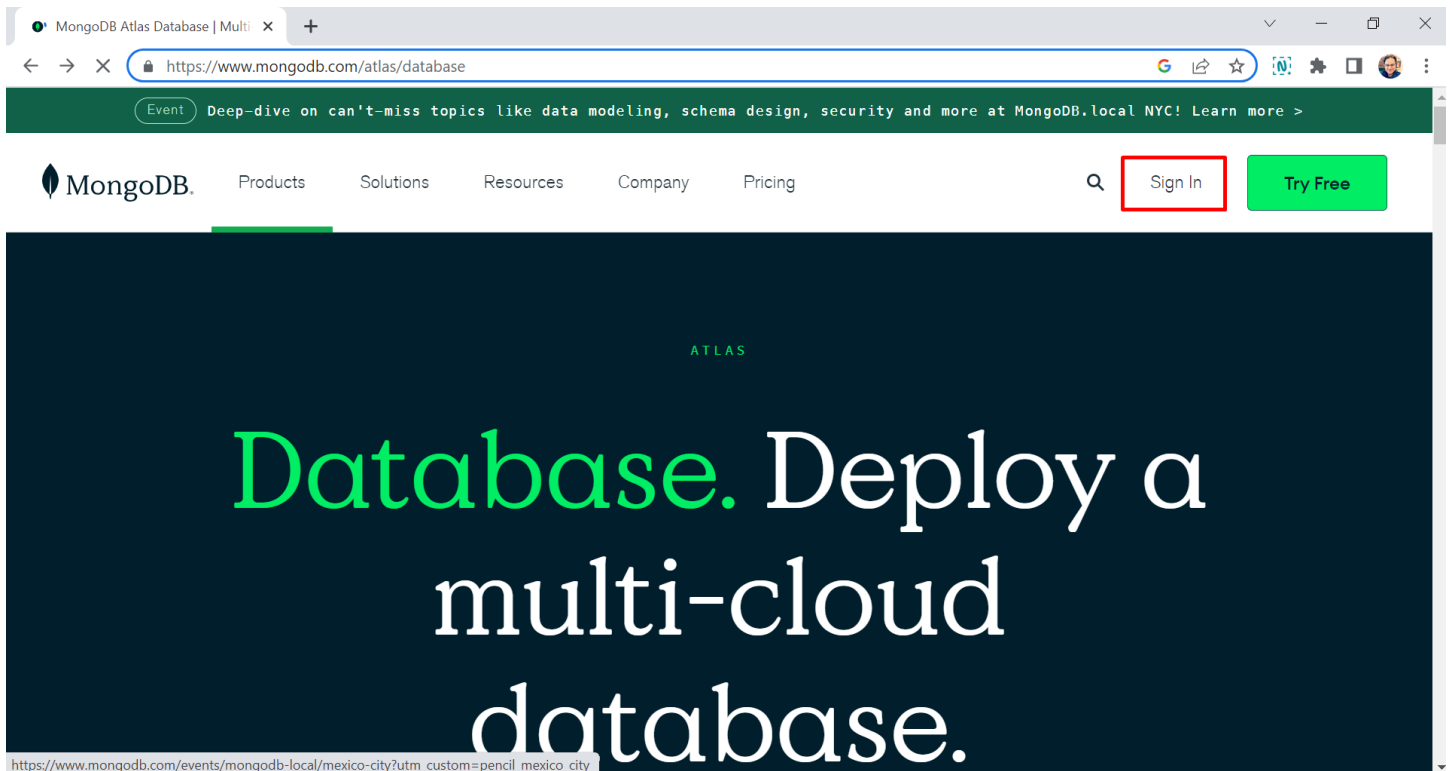


The screenshot shows the Visual Studio Code interface with a file explorer on the left showing the project structure: MERN > backend > node_modules, package-lock.json, package.json, server.mjs, and frontend. The main editor displays the content of `server.mjs`:

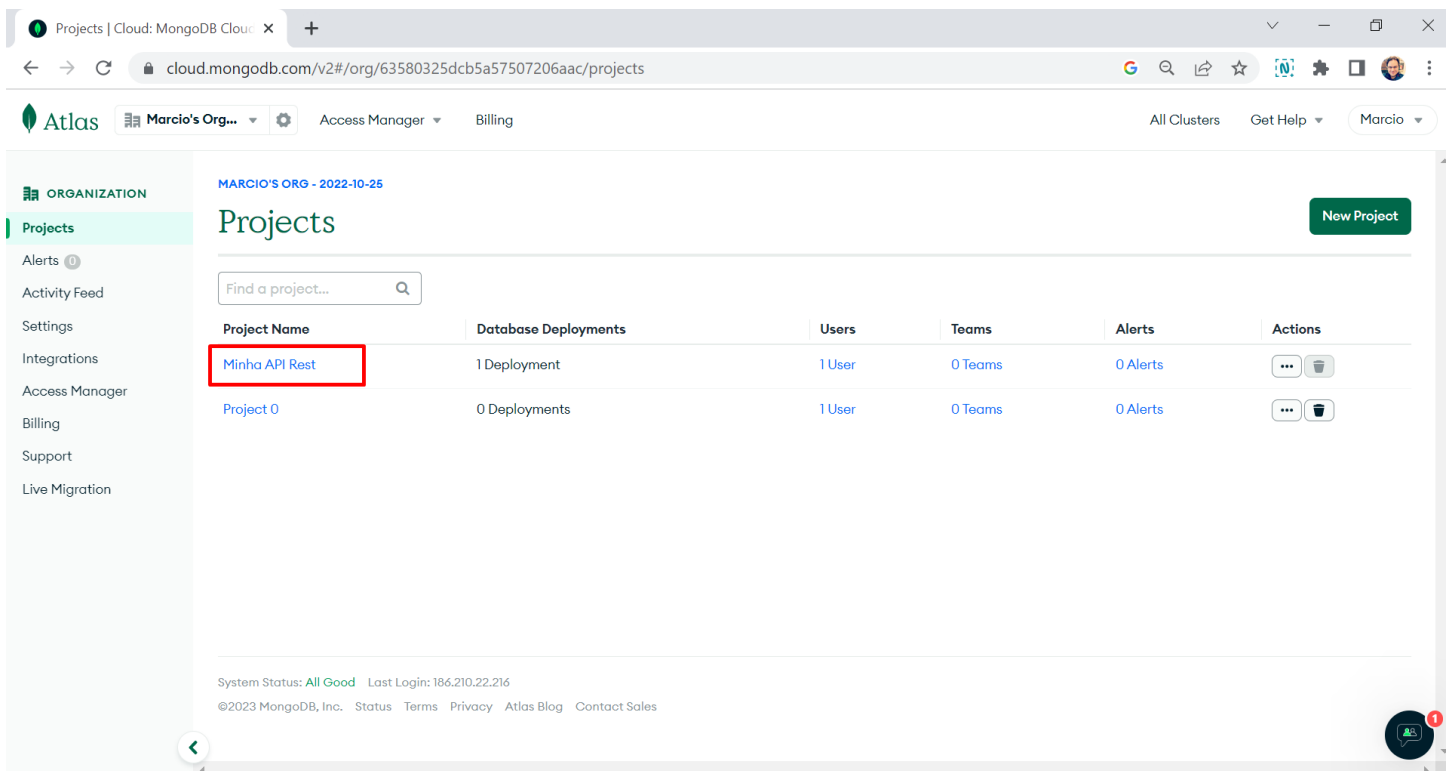
```
1 import express from "express";
2 import cors from "cors";
3
4 const PORT = process.env.PORT || 5050;
5 const app = express();
6
7 app.use(cors());
8 app.use(express.json());
9
10 app.listen(PORT, () => {
11   console.log(`Server is running on port: ${PORT}`);
12 });
```

At the bottom, the terminal window shows the command `node server.mjs` being executed in the backend directory, with the output: `Server is running on port: 5050`.

11 – Vamos agora conectar nosso banco MongoDB, faça login no Atlas: <https://www.mongodb.com/atlas/database>



12 – Após o login, vá até seus projetos. Vamos utilizar o cluster feito na última aula, caso não tenha volte até a aula anterior e siga os passos de criação de New Project. Clique em Minha API Rest ou no projeto já existente que tiver.



13 – Clique em Connect.

Database Deployments | Cloud: 1 x +

cloud.mongodb.com/v2/6463c2a911451523b69c6986#/clusters

Atlas Marcio's Org... Access Manager Billing All Clusters Get Help Marcio

Minha API Rest Data Services App Services Charts

DEPLOYMENT Database Data Lake PREVIEW SERVICES Triggers Data API Data Federation Search SECURITY Quickstart Backup Database Access Network Access Advanced New On Atlas 2

MARCIO'S ORG - 2022-10-25 > MINHA API REST

Database Deployments

Find a database deployment...

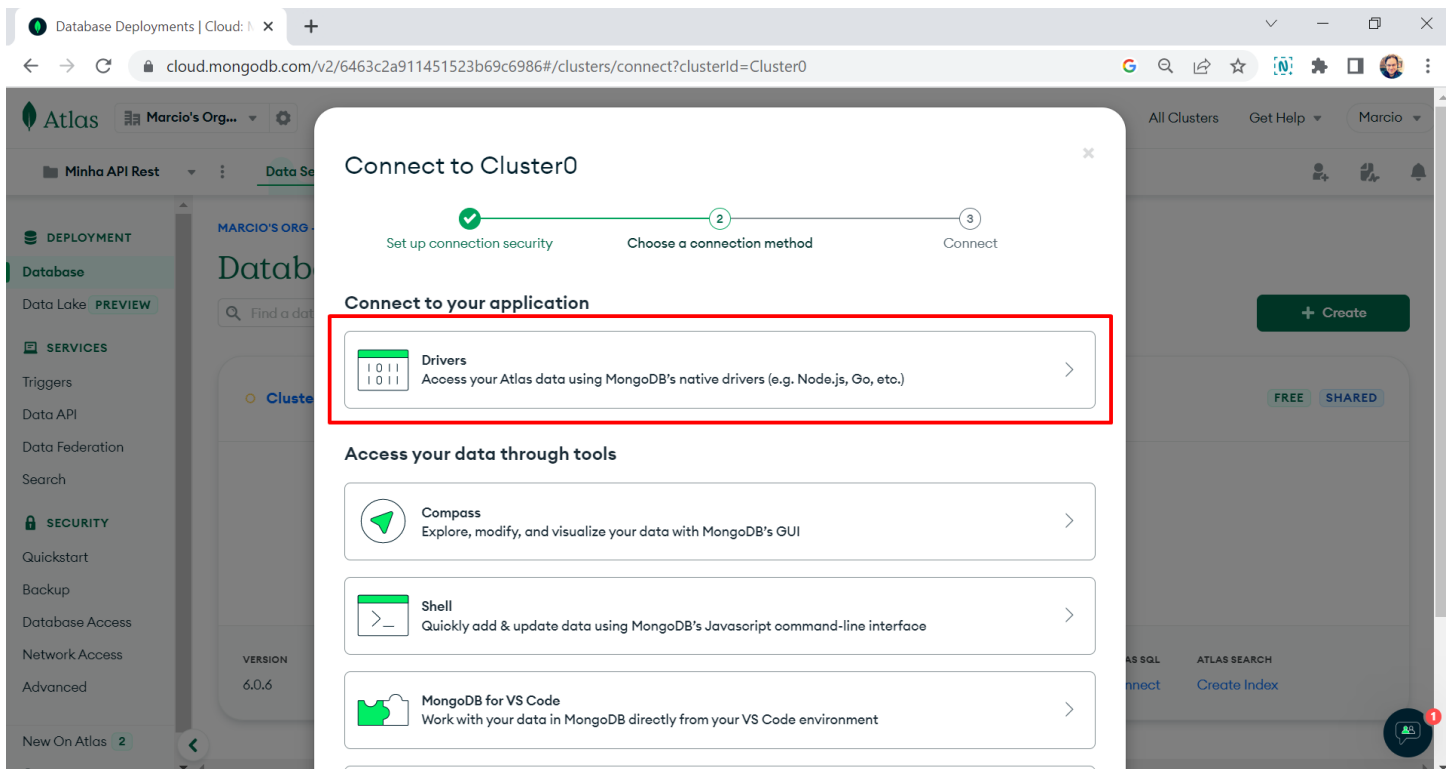
+ Create

Cluster0 Connect View Monitoring Browse Collections ... FREE SHARED

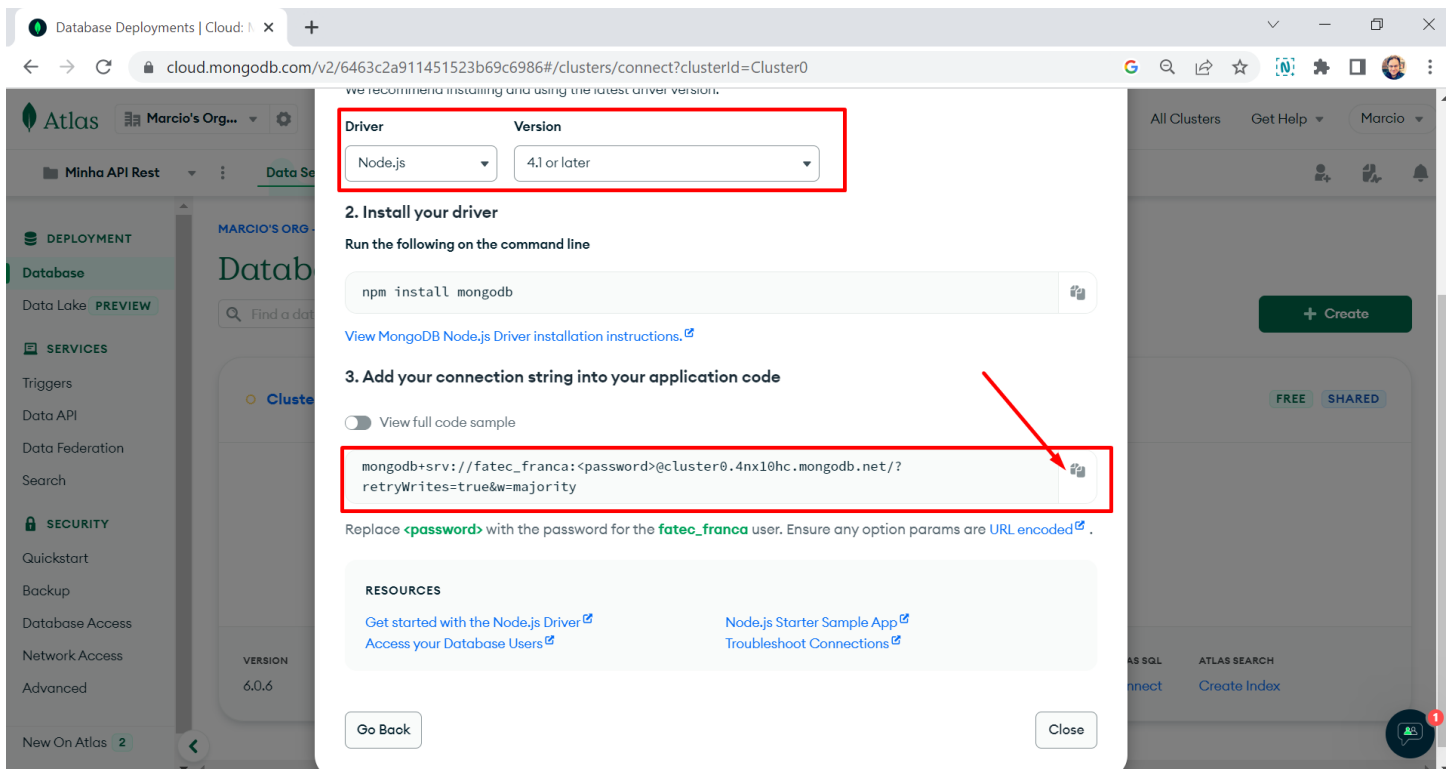
Monitoring for Cluster0 is Paused
Monitoring will automatically resume when you connect to your cluster.
[Visit the documentation](#) for more info.

VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
6.0.6	AWS / N. Virginia (us-east-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	Create Index

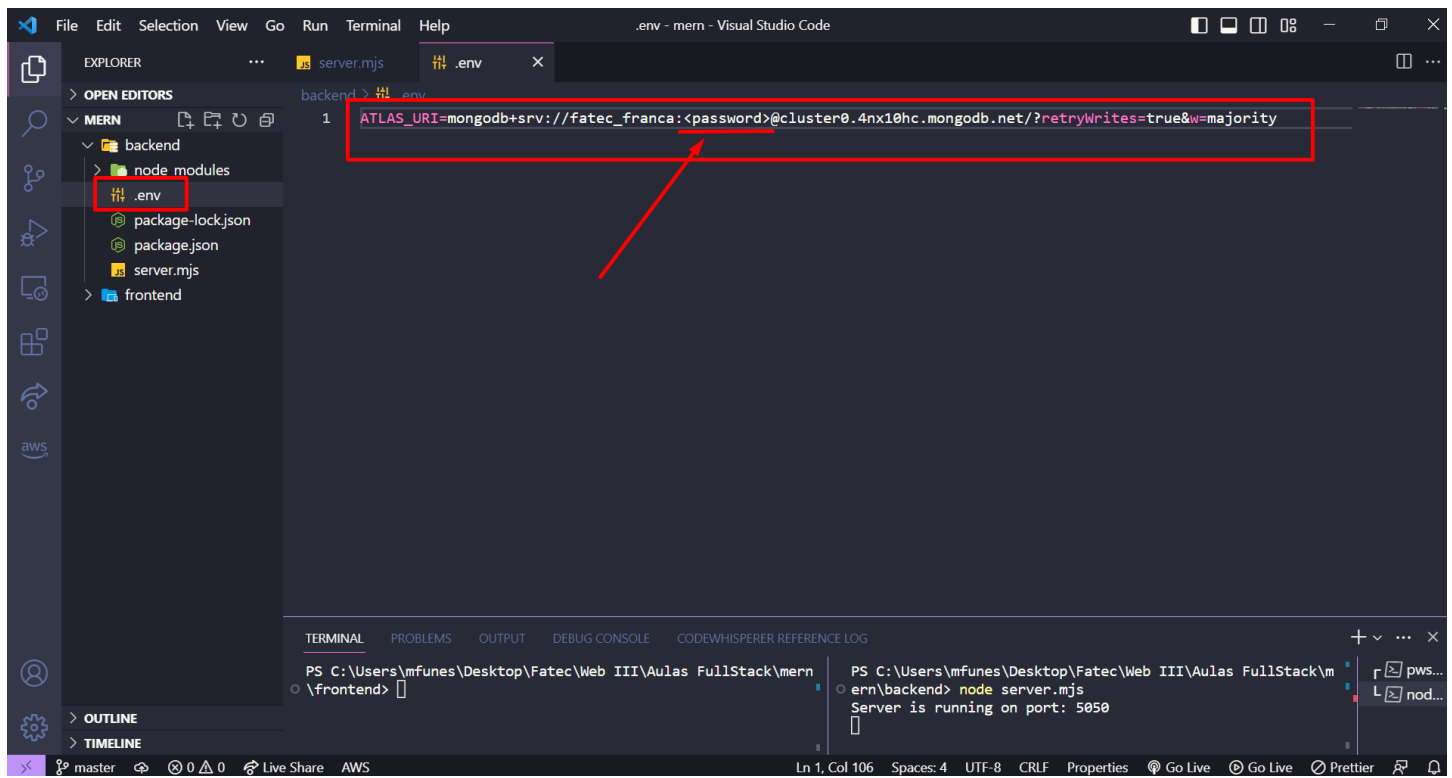
14 – Clique em Drivers.



15 – Confira se o driver está para Node.js e copie a string de conexão.

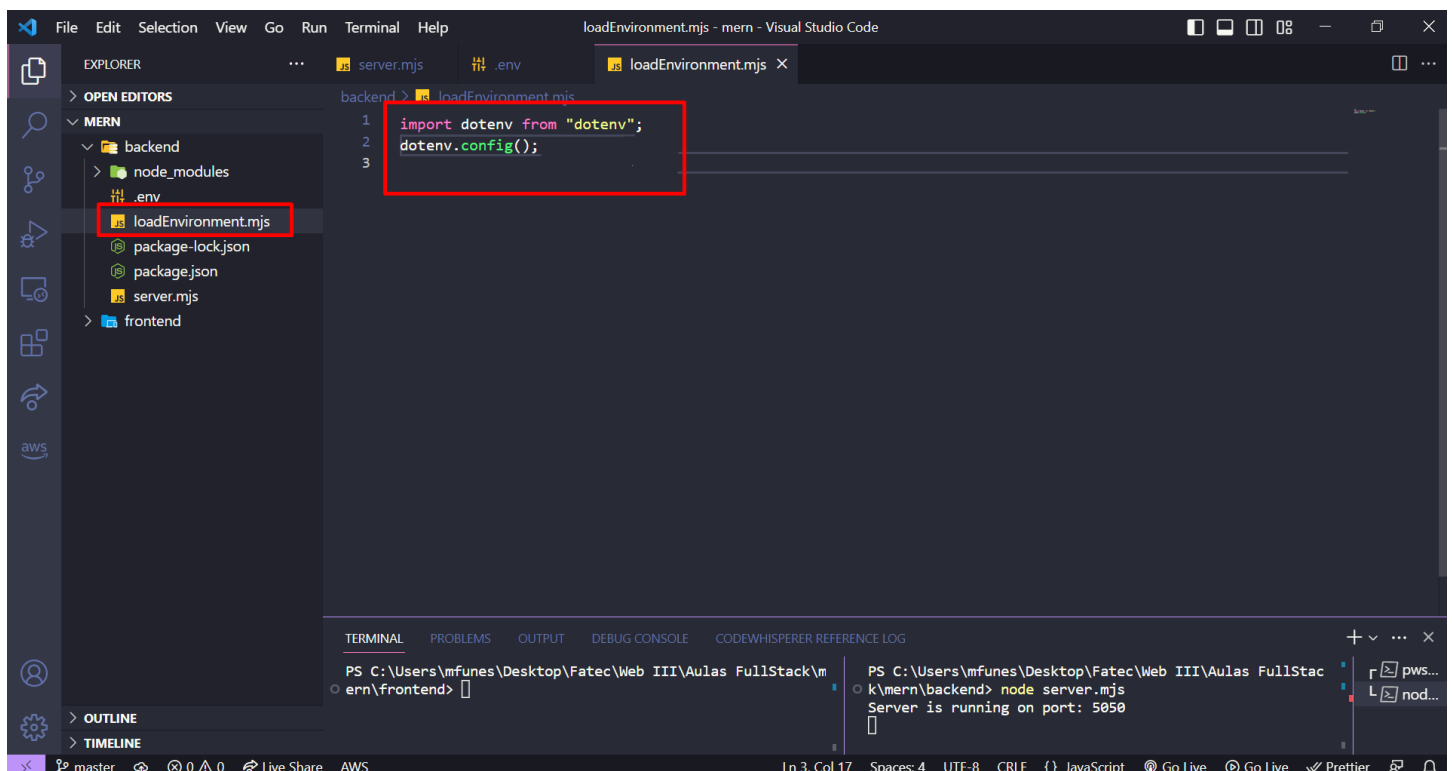


16 – De volta ao nosso projeto, na pasta backend crie um arquivo chamado .env e dentro coloque sua string desse modo. Não esqueça de mudar sua semana no item <password>.



```
ATLAS_URI=mongodb+srv://fatec_franca:<password>@cluster0.4nx10hc.mongodb.net/?retryWrites=true&w=majority
```

17 – Crie um novo arquivo no backend chamado loadEnvironment.mjs e coloque as importações abaixo:



```
import dotenv from "dotenv";
dotenv.config();
```

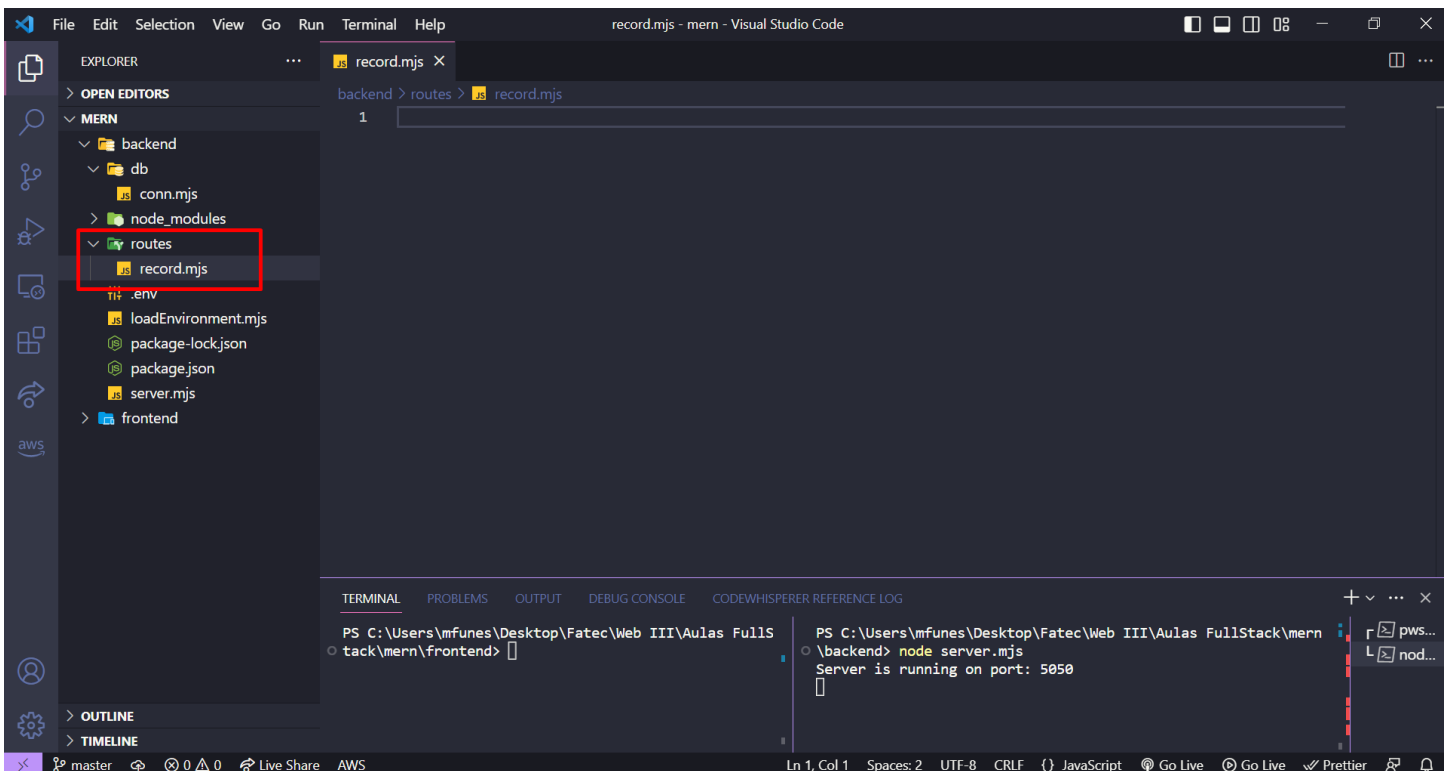
18 – Dentro do backend, crie uma pasta chamada db e dentro um arquivo chamado conn.mjs. Esse arquivo será responsável por uma abordagem mais moderna de conexão com o MongoDB.

The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The 'MERN' project is expanded, showing a 'backend' folder. Inside 'backend', a new folder named 'db' has been created, and inside it, a file named 'conn.mjs' is highlighted with a red box. The main editor area shows the code for 'conn.mjs', which is also highlighted with a red box. The code is as follows:

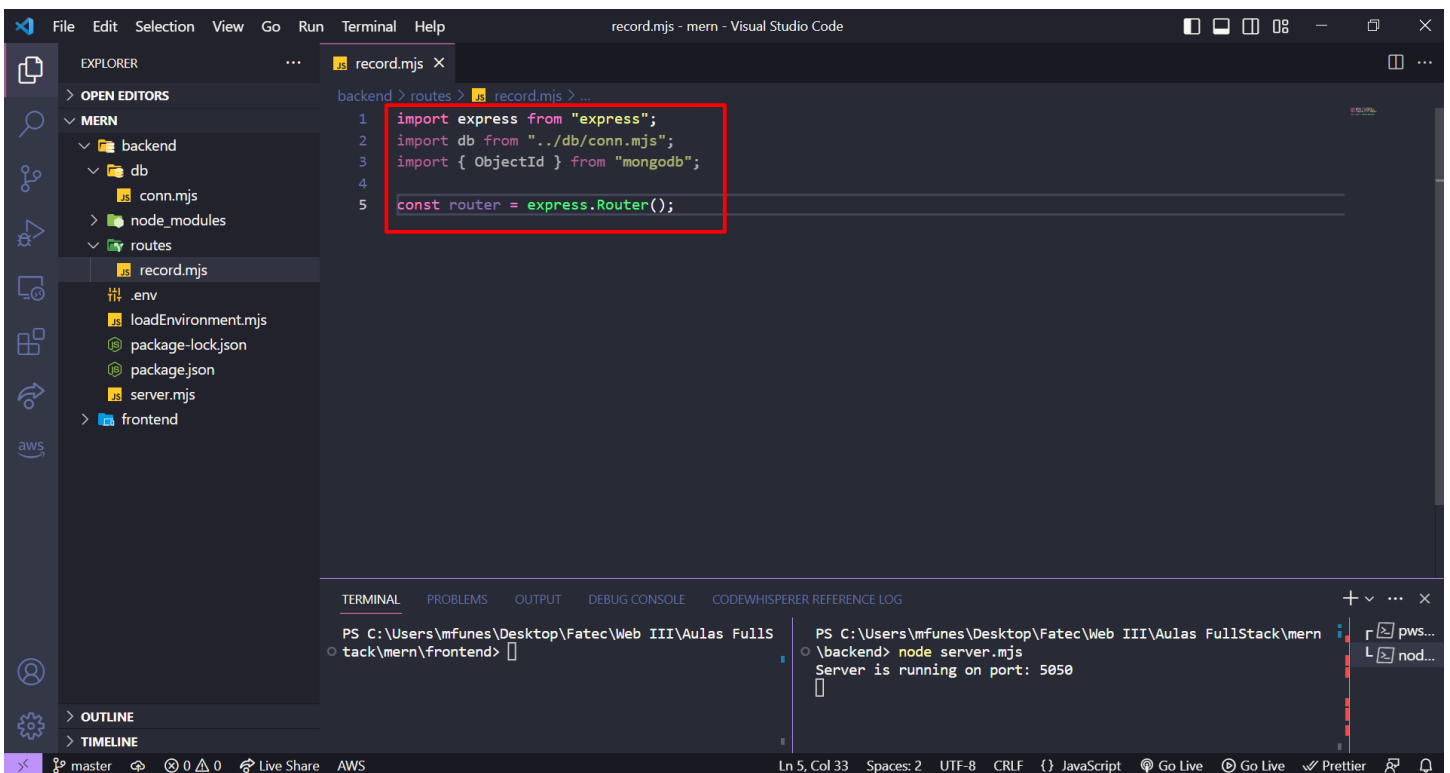
```
1 import { MongoClient } from "mongodb";
2
3 const connectionString = process.env.ATLAS_URI || "";
4
5 const client = new MongoClient(connectionString);
6
7 let conn;
8 try {
9   conn = await client.connect();
10 } catch(e) {
11   console.error(e);
12 }
13
14 let db = conn.db("");
15
16 export default db;
```

The bottom of the screen shows the Terminal panel with the command prompt at 'PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern\frontend>'. The output shows the server running on port 5050.

19 – Vamos agora configurar nossas rotas. Dentro de backend, crie uma pasta chamada routes e dentro um arquivo chamado record.mjs.



20 – Vamos começar fazendo as importações iniciais para que nossa API consiga ser utilizada em comunicação com o MongoDB.



21 – Configuração do GET da aplicação.

```
File Edit Selection View Go Run Terminal Help
record.mjs - mern - Visual Studio Code

EXPLORER
OPEN EDITORS
MERN
  backend
    db
      conn.mjs
    node_modules
    routes
      record.mjs
      env
      loadEnvironment.mjs
      package-lock.json
      package.json
      server.mjs
  frontend

record.mjs
1 import express from "express";
2 import db from "../db/conn.mjs";
3 import { ObjectId } from "mongodb";
4
5 const router = express.Router();
6
7 // GET - Lista todos os dados
8 router.get("/", async (req, res) => {
9   let collection = await db.collection("records");
10  let results = await collection.find({}).toArray();
11  res.send(results).status(200);
12 });
13
```

```
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
  o tack\mern\frontend>

PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
  o \backend> node server.mjs
Server is running on port: 5050
```

22 - Configuração do GET para listar dados por ID.

```
File Edit Selection View Go Run Terminal Help
record.mjs - mern - Visual Studio Code

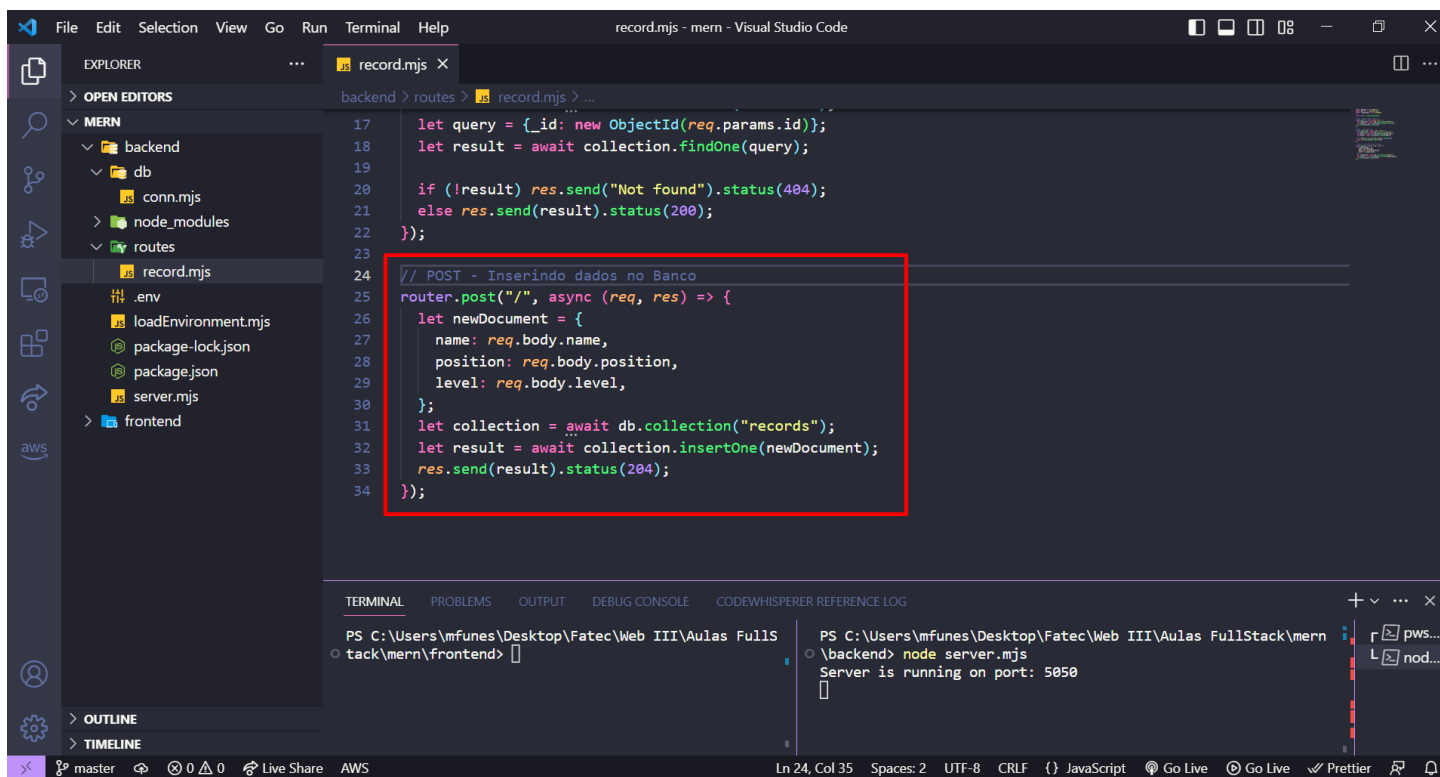
EXPLORER
OPEN EDITORS
MERN
  backend
    db
      conn.mjs
    node_modules
    routes
      record.mjs
      env
      loadEnvironment.mjs
      package-lock.json
      package.json
      server.mjs
  frontend

record.mjs
6
7 // GET - Lista todos os dados
8 router.get("/", async (req, res) => {
9   let collection = await db.collection("records");
10  let results = await collection.find({}).toArray();
11  res.send(results).status(200);
12 });
13
14 // GET - Lista por ID
15 router.get("/:id", async (req, res) => {
16   let collection = await db.collection("records");
17   let query = {_id: new ObjectId(req.params.id)};
18   let result = await collection.findOne(query);
19
20   if (!result) res.send("Not found").status(404);
21   else res.send(result).status(200);
22 });
```

```
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
  o tack\mern\frontend>

PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
  o \backend> node server.mjs
Server is running on port: 5050
```

23 - Configuração do POST para inserir dados.



```
File Edit Selection View Go Run Terminal Help
record.mjs - mern - Visual Studio Code

EXPLORER
OPEN EDITORS
MERN
  backend
    db
      conn.mjs
    node_modules
    routes
      record.mjs
    .env
    loadEnvironment.mjs
    package-lock.json
    package.json
    server.mjs
  frontend

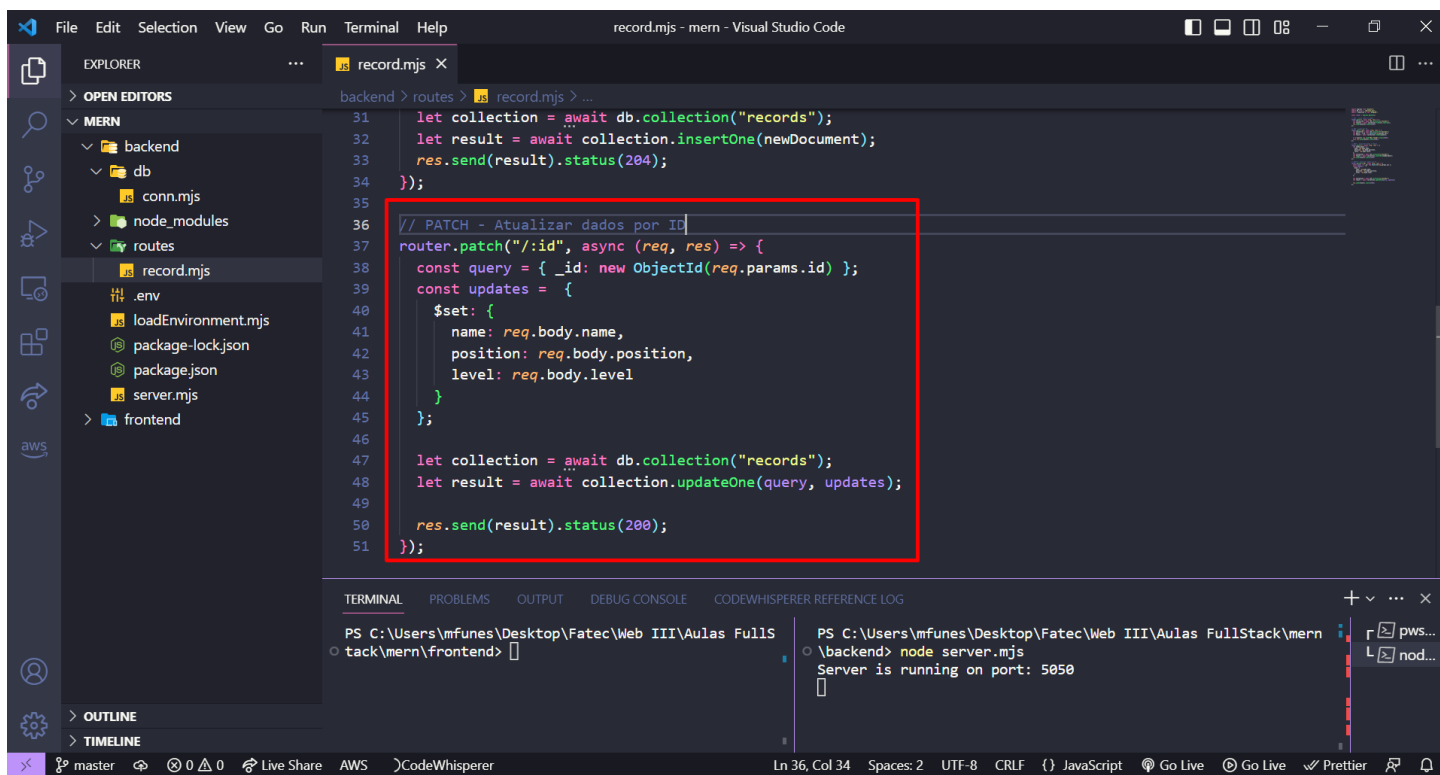
OUTLINE
TIMELINE

record.mjs
17 let query = { _id: new ObjectId(req.params.id)};
18 let result = await collection.findOne(query);
19
20 if (!result) res.send("Not found").status(404);
21 else res.send(result).status(200);
22 });
23
24 // POST - Inserindo dados no Banco
25 router.post("/", async (req, res) => {
26   let newDocument = {
27     name: req.body.name,
28     position: req.body.position,
29     level: req.body.level,
30   };
31   let collection = await db.collection("records");
32   let result = await collection.insertOne(newDocument);
33   res.send(result).status(204);
34 });

TERMINAL
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o tack\mern\frontend>

PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o \backend> node server.mjs
Server is running on port: 5050
```

24 – Configuração para a atualização de dados via PATCH.



```
File Edit Selection View Go Run Terminal Help
record.mjs - mern - Visual Studio Code

EXPLORER
OPEN EDITORS
MERN
  backend
    db
      conn.mjs
    node_modules
    routes
      record.mjs
    .env
    loadEnvironment.mjs
    package-lock.json
    package.json
    server.mjs
  frontend

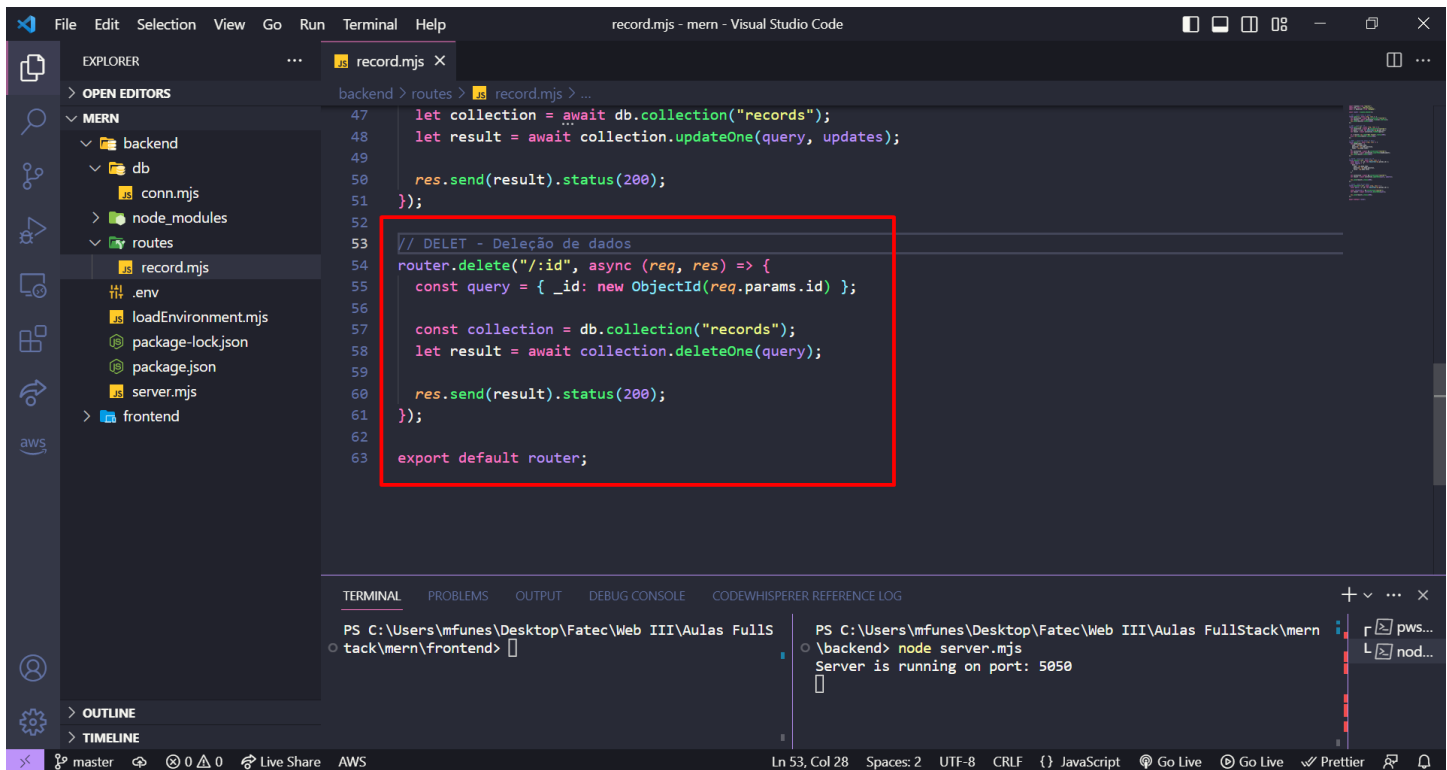
OUTLINE
TIMELINE

record.mjs
31 let collection = await db.collection("records");
32 let result = await collection.insertOne(newDocument);
33 res.send(result).status(204);
34 });
35
36 // PATCH - Atualizar dados por ID
37 router.patch("/:id", async (req, res) => {
38   const query = { _id: new ObjectId(req.params.id) };
39   const updates = {
40     $set: {
41       name: req.body.name,
42       position: req.body.position,
43       level: req.body.level
44     }
45   };
46
47   let collection = await db.collection("records");
48   let result = await collection.updateOne(query, updates);
49
50   res.send(result).status(200);
51 });

TERMINAL
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o tack\mern\frontend>

PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o \backend> node server.mjs
Server is running on port: 5050
```

25 – Por fim, nosso DELETE e exportação das rotas.



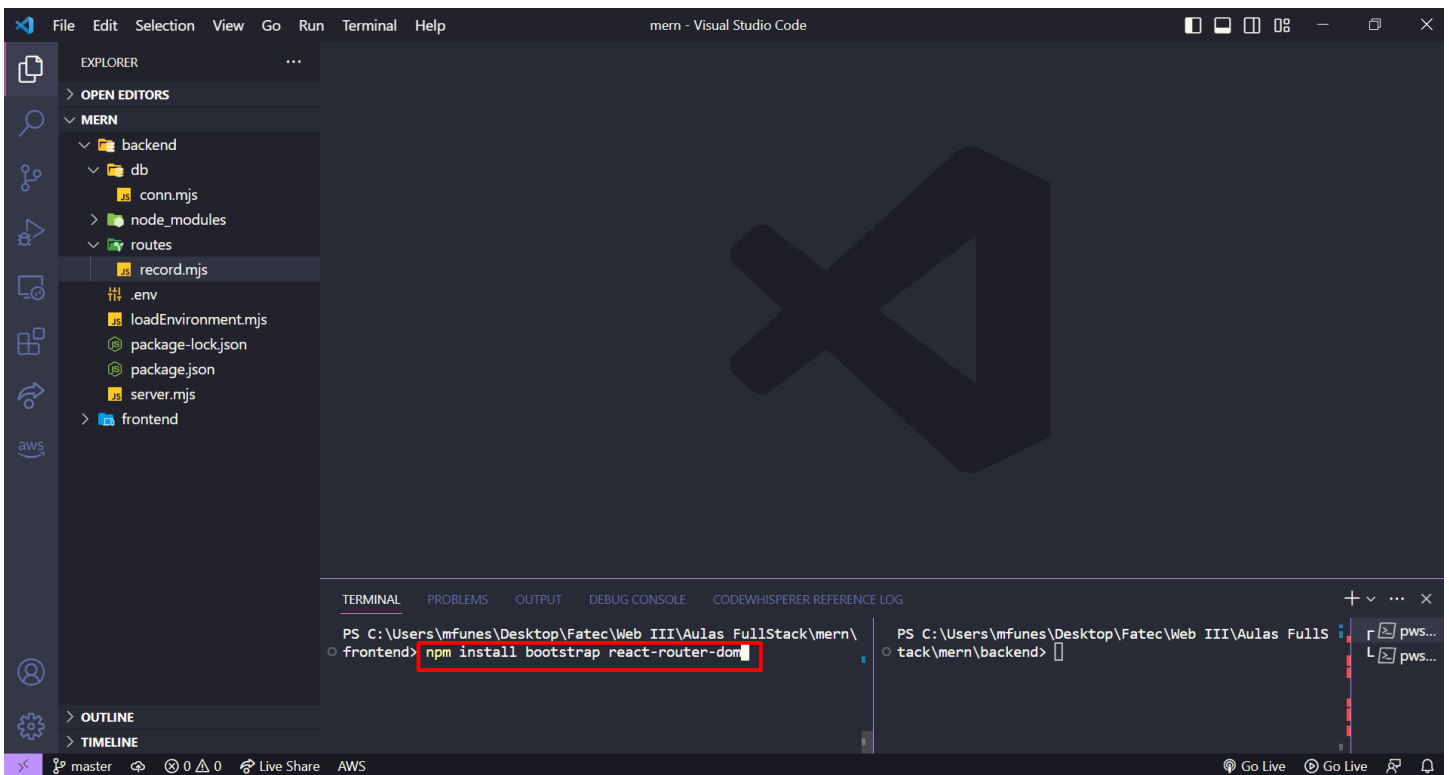
```
47 let collection = await db.collection("records");
48 let result = await collection.updateOne(query, updates);
49
50 res.send(result).status(200);
51 });
52
53 // DELET - Deleção de dados
54 router.delete("/:id", async (req, res) => {
55   const query = { _id: new ObjectId(req.params.id) };
56
57   const collection = db.collection("records");
58   let result = await collection.deleteOne(query);
59
60   res.send(result).status(200);
61 });
62
63 export default router;
```

Terminal output:

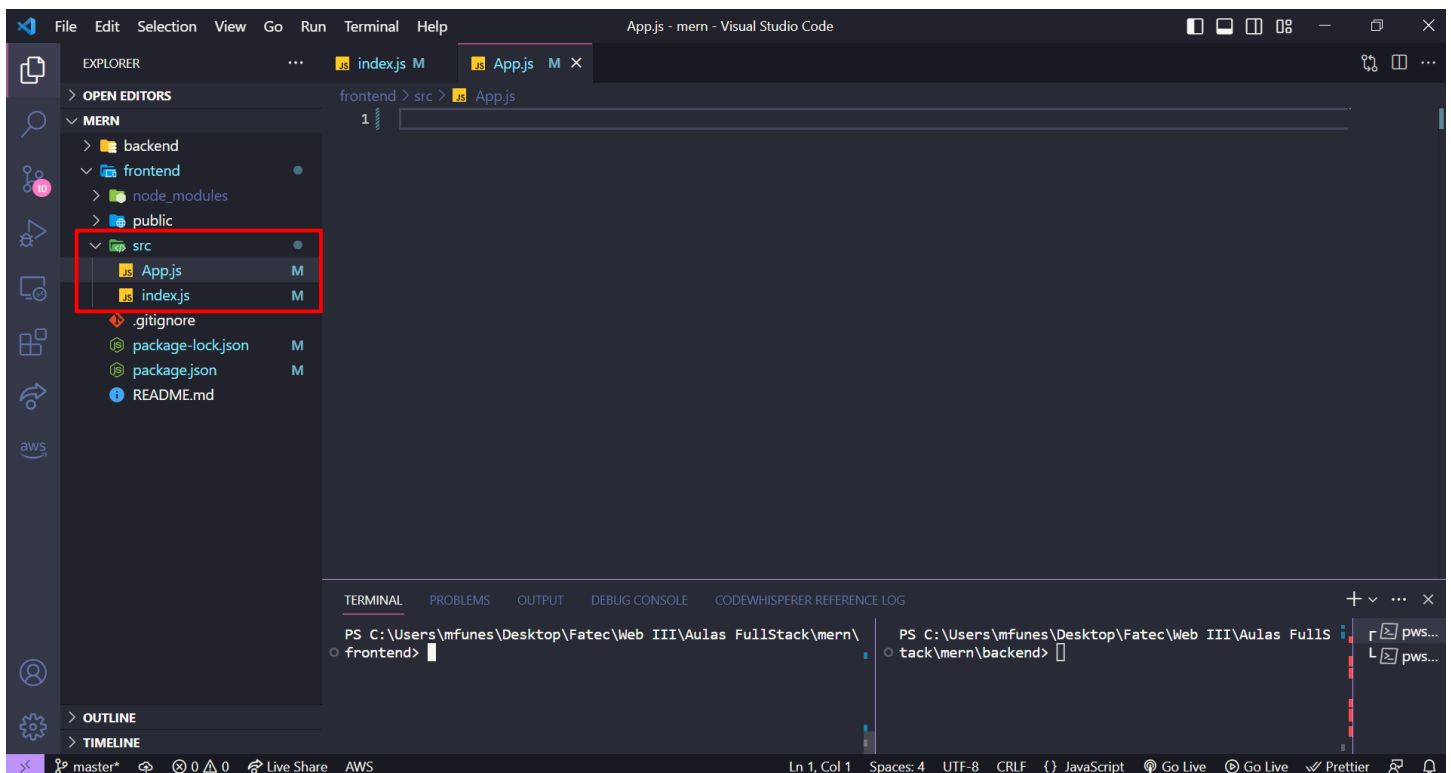
```
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o tack\mern\frontend>
PS C:\Users\mfunes\Desktop\Fatec\Web III\Aulas FullStack\mern
o \backend> node server.mjs
Server is running on port: 5050
```

FRONTEND

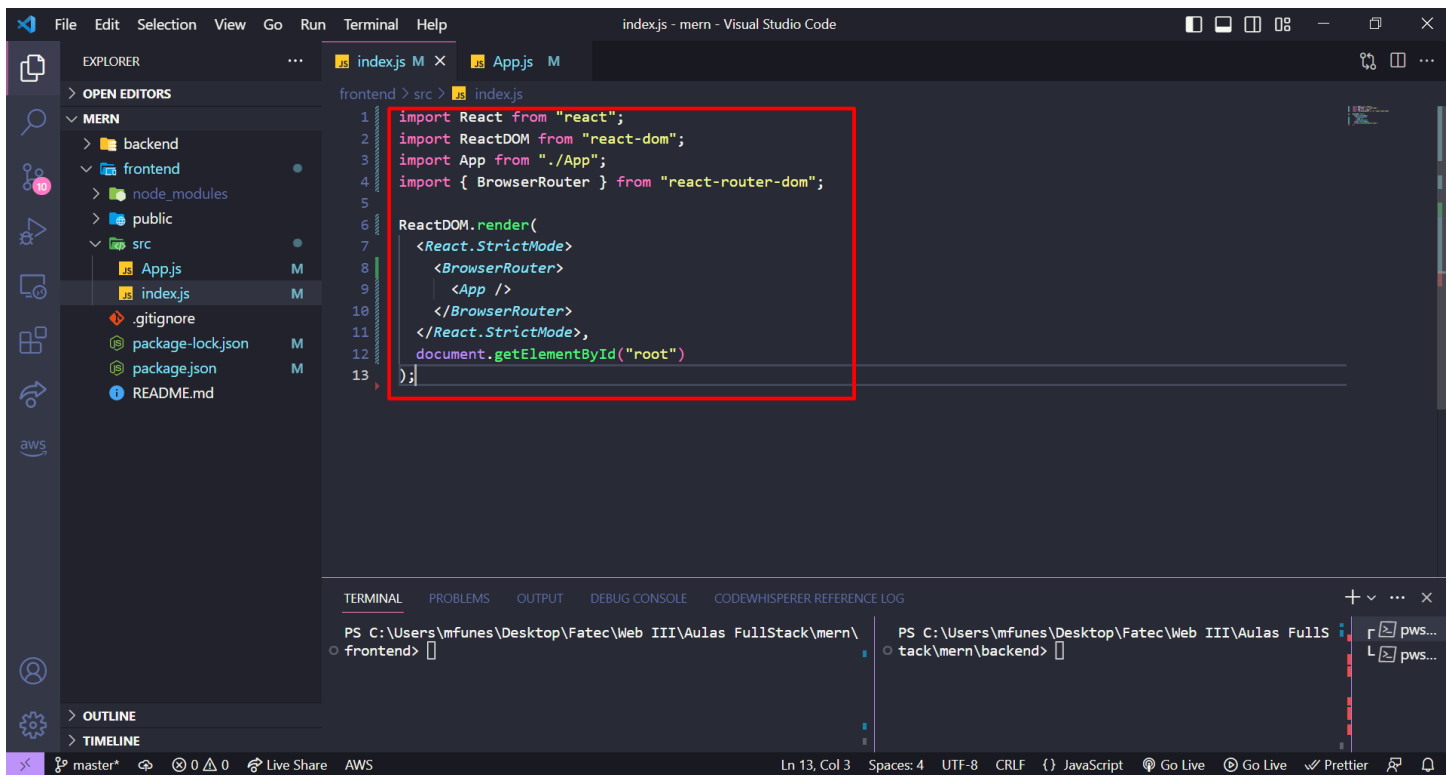
26 – Agora que finalizamos nosso backend vamos configurar o frontend da aplicação. Vá até o terminal do frontend e vamos instalar algumas dependências, digite: “npm install bootstrap react-router-dom”



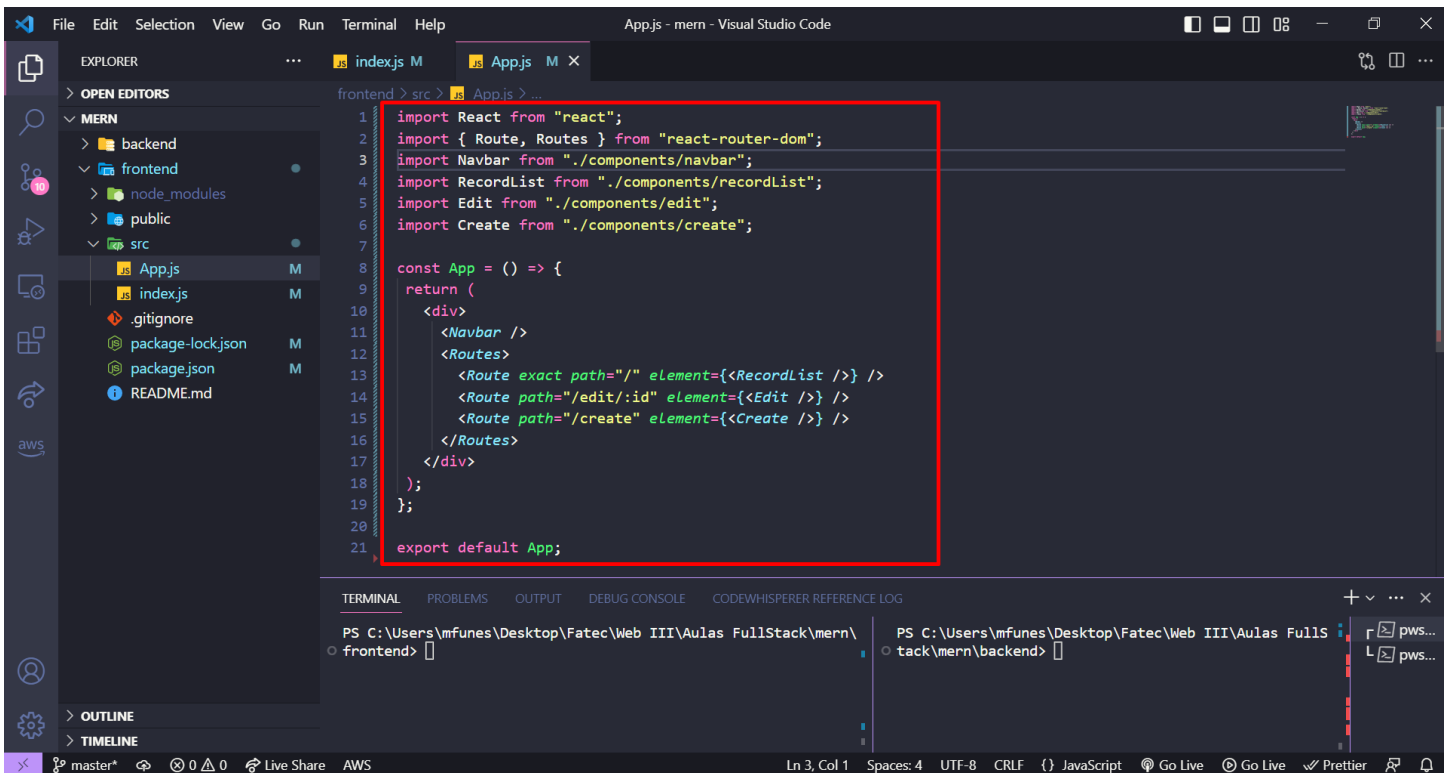
27 – Apague todos os arquivos de dentro da pasta src, depois crie um arquivo chamado App.js e outro index.js.



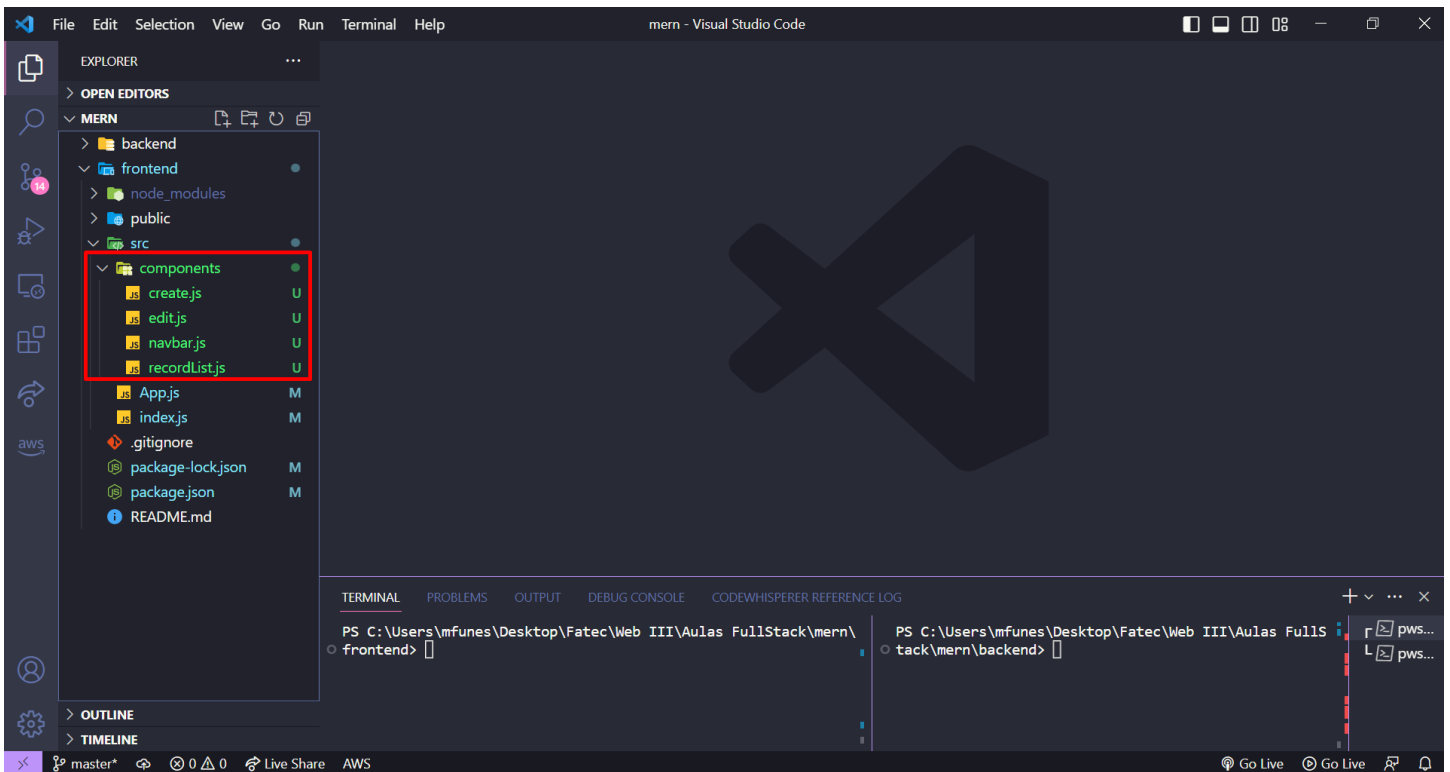
28 – Dentro de index.js vamos fazer a configuração padrão de um index.js inicial.



29 – E em App.js vamos colocar aqui os componentes que ainda vamos criar, componentes esses que farão parte da nosso front.



30 – Dentro de src, crie uma pasta chamada components e 4 arquivos como mostrado abaixo:



31 – create.js

import React, { useState } from "react";


```
import { useNavigate } from "react-router";

export default function Create() {
  const [form, setForm] = useState({
    name: "",
    position: "",
    level: "",
  });

  const navigate = useNavigate();

  function updateForm(value) {
    return setForm((prev) => {
      return { ...prev, ...value };
    });
  }

  async function onSubmit(e) {
    e.preventDefault();

    const newPerson = { ...form };

    await fetch("http://localhost:5050/record", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
```

```

    body: JSON.stringify(newPerson),
  })
  .catch(error => {
    window.alert(error);
    return;
  });

  setForm({ name: "", position: "", level: "" });
  navigate("/");
}

return (
  <div>
    <h3>Create New Record</h3>
    <form onSubmit={onSubmit}>
      <div className="form-group">
        <label htmlFor="name">Name</label>
        <input
          type="text"
          className="form-control"
          id="name"
          value={form.name}
          onChange={(e) => updateForm({ name: e.target.value })}
        />
      </div>

```

```
<div className="form-group">
  <label htmlFor="position">Position</label>
  <input
    type="text"
    className="form-control"
    id="position"
    value={form.position}
    onChange={(e) => updateForm({ position: e.target.value })}
  />
</div>
```

```
<div className="form-group">
  <div className="form-check form-check-inline">
    <input
      className="form-check-input"
      type="radio"
      name="positionOptions"
      id="positionIntern"
      value="Intern"
      checked={form.level === "Intern"}
      onChange={(e) => updateForm({ level: e.target.value })}
    />
    <label      htmlFor="positionIntern"
className="form-check-label">Intern</label>
  </div>
  <div className="form-check form-check-inline">
```

```
<input
  className="form-check-input"
  type="radio"
  name="positionOptions"
  id="positionJunior"
  value="Junior"
  checked={form.level === "Junior"}
  onChange={(e) => updateForm({ level: e.target.value })}
/>
```

```
<label      htmlFor="positionJunior"
```

```
className="form-check-label">Junior</label>
```

```
</div>
```

```
<div className="form-check form-check-inline">
```

```
<input
  className="form-check-input"
  type="radio"
  name="positionOptions"
  id="positionSenior"
  value="Senior"
  checked={form.level === "Senior"}
  onChange={(e) => updateForm({ level: e.target.value })}
/>
```

```
<label      htmlFor="positionSenior"
```

```
className="form-check-label">Senior</label>
```

```
</div>
```

```
</div>
```

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

```
  <input
```

```
    type="submit"
```

```
    value="Create person"
```

```
    className="btn btn-primary"
```

```
  />
```

```
</div>
```

```
</form>
```

```
</div>
```

```
);
```

```
}
```

32 – edit.js

```
import React, { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router";

export default function Edit() {
  const [form, setForm] = useState({
    name: "",
    position: "",
    level: "",
    records: [],
  });

  const params = useParams();
  const navigate = useNavigate();

  useEffect(() => {
    async function fetchData() {
      const id = params.id.toString();

      const response = await fetch(`http://localhost:5050/record/${params.id.toString()}`);

      if (!response.ok) {
        const message = `An error has occurred: ${response.statusText}`;
        window.alert(message);
        return;
      }
    }
  });
}
```

```
const record = await response.json();
if (!record) {
  window.alert(`Record with id ${id} not found`);
  navigate("/");
  return;
}

setForm(record);
}

fetchData();

return;
}, [params.id, navigate]);
```

// These methods will update the state properties.

```
function updateForm(value) {
  return setForm((prev) => {
    return { ...prev, ...value };
  });
}
```

```
async function onSubmit(e) {
  e.preventDefault();
  const editedPerson = {
    name: form.name,
```

```
    position: form.position,  
    level: form.level,  
  };
```

```
// This will send a post request to update the data in the database.
```

```
await fetch(`http://localhost:5050/record/${params.id}`, {  
  method: "PATCH",  
  body: JSON.stringify(editedPerson),  
  headers: {  
    'Content-Type': 'application/json'  
  },  
});
```

```
  navigate("/");  
}
```

```
// This following section will display the form that takes input from the user to  
update the data.
```

```
return (  
  <div>  
    <h3>Update Record</h3>  
    <form onSubmit={onSubmit}>  
      <div className="form-group">  
        <label htmlFor="name">Name: </label>  
        <input  
          type="text"
```



```
      className="form-control"
      id="name"
      value={form.name}
      onChange={(e) => updateForm({ name: e.target.value })}
    />
  </div>
  <div className="form-group">
    <label htmlFor="position">Position: </label>
    <input
      type="text"
      className="form-control"
      id="position"
      value={form.position}
      onChange={(e) => updateForm({ position: e.target.value })}
    />
  </div>
  <div className="form-group">
    <div className="form-check form-check-inline">
      <input
        className="form-check-input"
        type="radio"
        name="positionOptions"
        id="positionIntern"
        value="Intern"
        checked={form.level === "Intern"}
      />
    </div>
  </div>
```

```

        onChange={(e) => updateForm({ level: e.target.value })}
    />
                                <label      htmlFor="positionIntern"
className="form-check-label">Intern</label>
    </div>
    <div className="form-check form-check-inline">
        <input
            className="form-check-input"
            type="radio"
            name="positionOptions"
            id="positionJunior"
            value="Junior"
            checked={form.level === "Junior"}
            onChange={(e) => updateForm({ level: e.target.value })}
        />
                                <label      htmlFor="positionJunior"
className="form-check-label">Junior</label>
    </div>
    <div className="form-check form-check-inline">
        <input
            className="form-check-input"
            type="radio"
            name="positionOptions"
            id="positionSenior"
            value="Senior"

```

```

        checked={form.level === "Senior"}
        onChange={(e) => updateForm({ level: e.target.value })}
    />
                                <label      htmlFor="positionSenior"
className="form-check-label">Senior</label>
    </div>
    </div>
    <br />

    <div className="form-group">
        <input
            type="submit"
            value="Update Record"
            className="btn btn-primary"
        />
    </div>
    </form>
    </div>
    );
}

```

33 – recordList.js

```
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";

const Record = (props) => (
  <tr>
    <td>{props.record.name}</td>
    <td>{props.record.position}</td>
    <td>{props.record.level}</td>
    <td>
      <Link className="btn btn-link" to={` /edit/${props.record._id}`}>Edit</Link> |
      <button className="btn btn-link"
        onClick={() => {
          props.deleteRecord(props.record._id);
        }}
      >
        Delete
      </button>
    </td>
  </tr>
);

export default function RecordList() {
  const [records, setRecords] = useState([]);

  // This method fetches the records from the database.
  useEffect(() => {
    async function getRecords() {
      const response = await fetch(`http://localhost:5050/record/`);
```

```
if (!response.ok) {  
  const message = `An error occurred: ${response.statusText}`;  
  window.alert(message);  
  return;  
}
```

```
const records = await response.json();  
setRecords(records);  
}
```

```
getRecords();
```

```
return;  
}, [records.length]);
```

```
// This method will delete a record
```

```
async function deleteRecord(id) {  
  await fetch(`http://localhost:5050/record/${id}`, {  
    method: "DELETE"  
  });  
}
```

```
const newRecords = records.filter((el) => el._id !== id);  
setRecords(newRecords);  
}
```

```
// This method will map out the records on the table
```

```
function recordList() {  
  return records.map((record) => {  
    return (  
      <Record  
        record={record}
```

```
      deleteRecord={() => deleteRecord(record._id)}
      key={record._id}
    />
  );
});
}
```

// This following section will display the table with the records of individuals.

```
return (
  <div>
    <h3>Record List</h3>
    <table className="table table-striped" style={{ marginTop: 20 }}>
      <thead>
        <tr>
          <th>Name</th>
          <th>Position</th>
          <th>Level</th>
          <th>Action</th>
        </tr>
      </thead>
      <tbody>{recordList()}</tbody>
    </table>
  </div>
);
}
```

34 – navbar.js

```
import React from "react";
```

```
// We import bootstrap to make our application look better.
```

```
import "bootstrap/dist/css/bootstrap.css";
```

```
// We import NavLink to utilize the react router.
```

```
import { NavLink } from "react-router-dom";
```

```
// Here, we display our Navbar
```

```
export default function Navbar() {
```

```
  return (
```

```
    <div>
```

```
      <nav className="navbar navbar-expand-lg navbar-light bg-light">
```

```
        <NavLink className="navbar-brand" to="/">
```

```
          </img>
```

```
        </NavLink>
```

```
        <button
```

```
          className="navbar-toggler"
```

```
          type="button"
```

```
          data-toggle="collapse"
```

```
          data-target="#navbarSupportedContent"
```

```
          aria-controls="navbarSupportedContent"
```

```
          aria-expanded="false"
```

```
          aria-label="Toggle navigation"
```

```
        >
```

```
          <span className="navbar-toggler-icon"></span>
```

</button>

<div className="collapse navbar-collapse" id="navbarSupportedContent">

<ul className="navbar-nav ml-auto">

<li className="nav-item">

<NavLink className="nav-link" to="/create">

Create Record

</NavLink>

</div>

</nav>

</div>

);

}