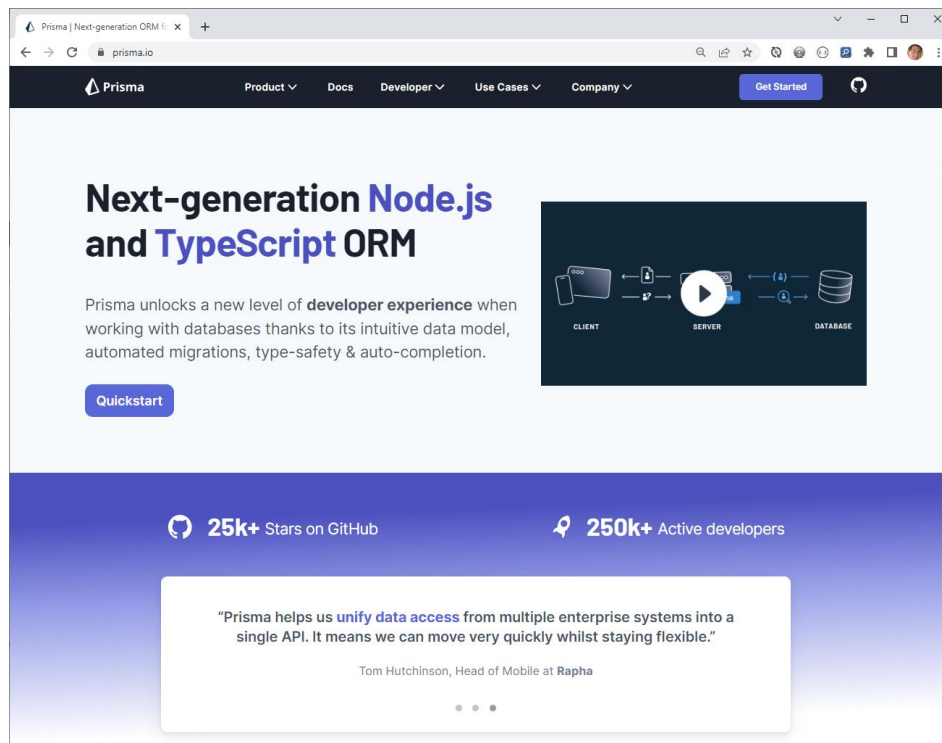


Lista 17a – *Back-end*: API REST com node.js e Persistência com o ORM Prisma

Esse roteiro utiliza o MySQL. É necessário, portanto, que ele esteja instalado e o servidor em execução.

A persistência será implementada com o auxílio do *framework* ORM [Prisma](#).



1. Crie uma pasta chamada *lista17* e dentro dela as pastas *back-end-prisma* e *front-end*. Dentro da pasta *back-end-prisma* crie outra pasta chamada *src* e dentro da pasta *src* crie o arquivo *server.js*. Certifique-se que os nomes dos arquivos e pastas estão corretos e que o arquivo *server.js* está realmente dentro da pasta *src*.

2. Abra o terminal e digite os comandos abaixo para criar o projeto node.js

```
cd .\back-end-prisma\  
npm init -y  
npm i express cors express-form-data mysql2  
npm i nodemon prisma --save-dev
```

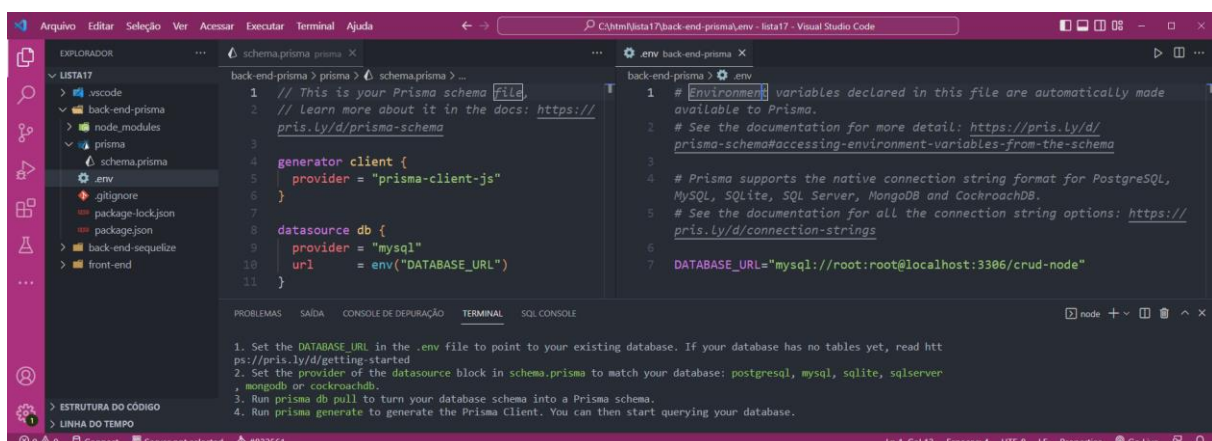
Acrescentar ao *package.json*

```
"scripts": {  
  "dev": "nodemon ./src/server.js",  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

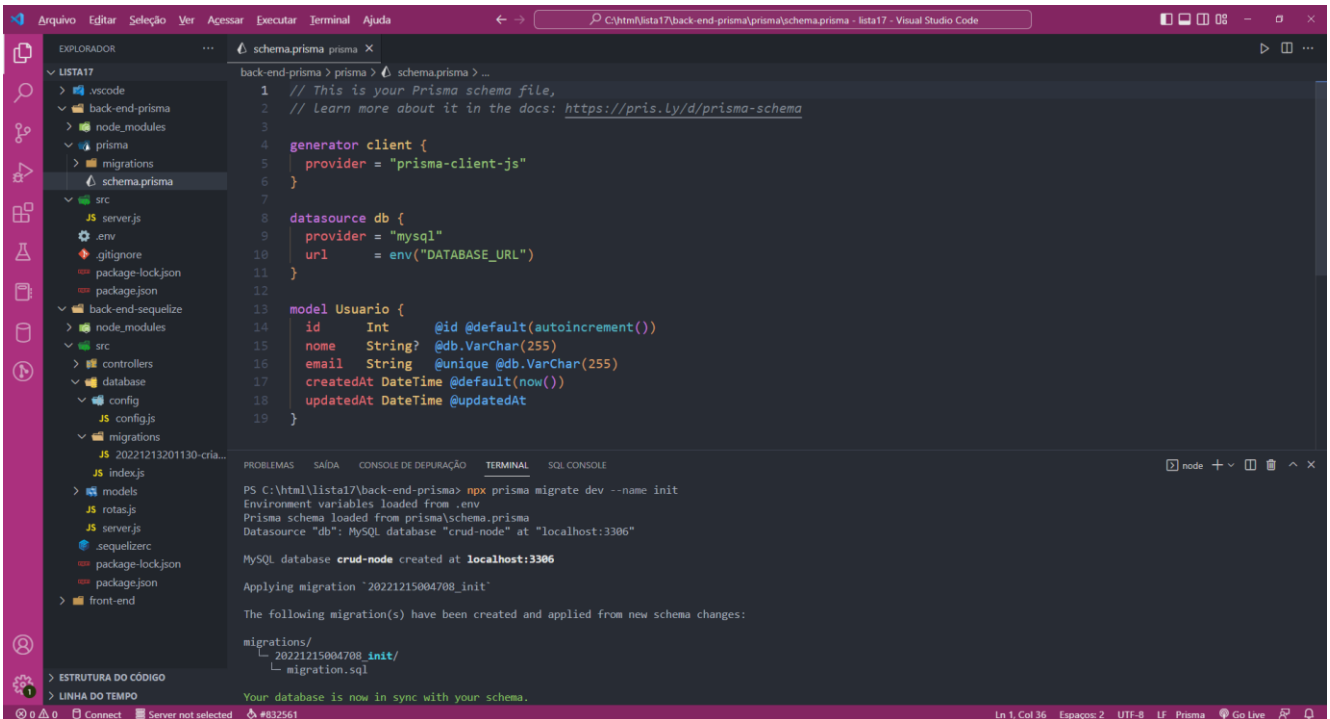
3. O próximo comando irá inicializar um projeto Prisma, criando as pastas e arquivos necessários para o *framework*.

- `npx prisma init`

Modifique os arquivos *schema.prisma* e *.env* como mostrado abaixo.



4. Preencha o modelo (descrição) da tabela no arquivo *schema.prisma* e após isso execute o comando a seguir:
- `npx prisma migrate dev --name init`



The screenshot shows the Visual Studio Code editor with the `schema.prisma` file open. The schema defines a `Usuario` model with fields `id`, `nome`, `email`, `createdAt`, and `updatedAt`. The terminal shows the execution of `npx prisma migrate dev --name init`, which successfully creates the MySQL database `crud-node` and applies the migration `20221215004708_init`.

```
// This is your Prisma schema file,
// Learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model Usuario {
  id          Int      @id @default(autoincrement())
  nome        String?  @db.VarChar(255)
  email       String   @unique @db.VarChar(255)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}
```

Terminal Output:

```
PS C:\html\lista17\back-end-prisma> npx prisma migrate dev --name init
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": MySQL database "crud-node" at "localhost:3306"

MySQL database crud-node created at localhost:3306

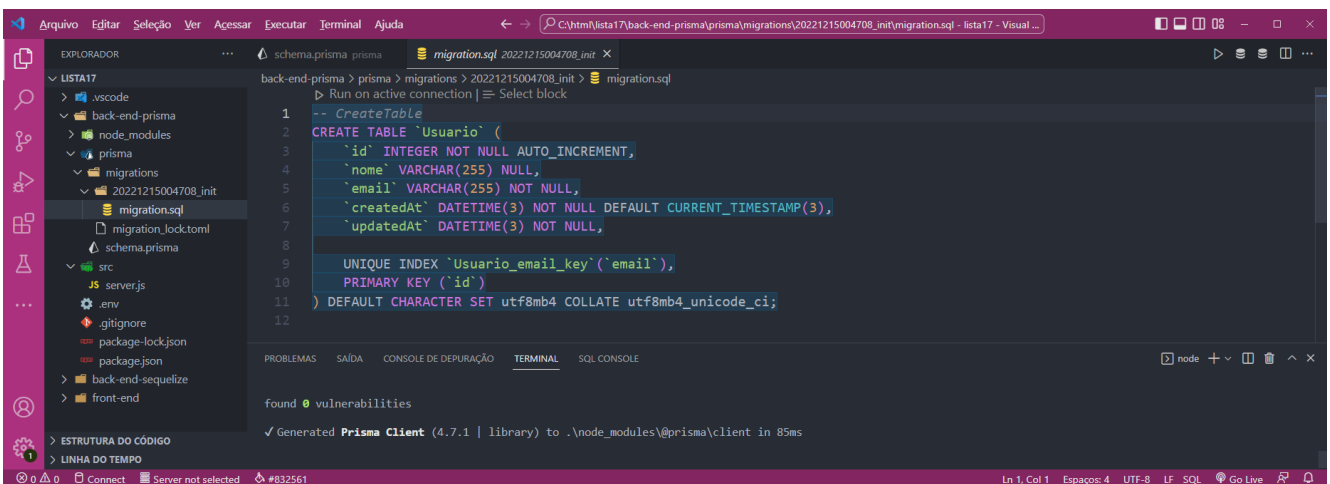
Applying migration `20221215004708_init`

The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20221215004708_init/
└─ migration.sql

Your database is now in sync with your schema.
```

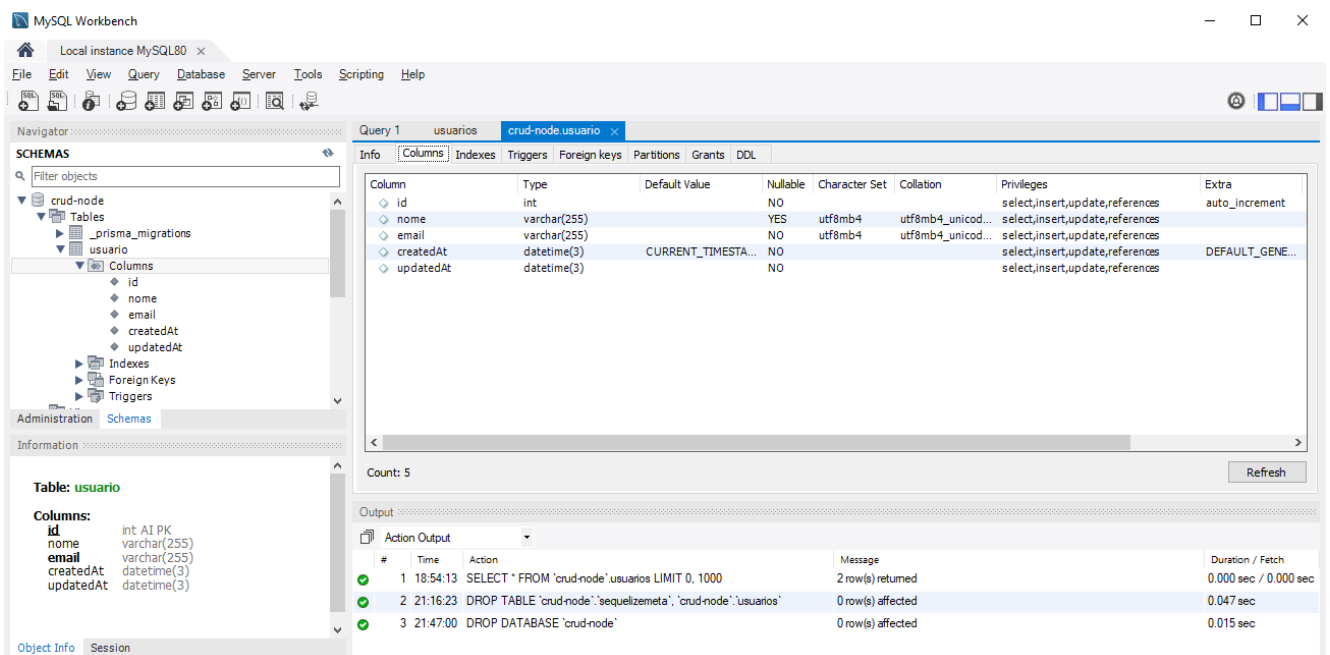
Abaixo, dentro da pasta *migrations*, pode ser visto o arquivo de migração que foi criado.



The screenshot shows the `migration.sql` file generated for the migration `20221215004708_init`. It contains SQL code to create the `Usuario` table with the specified schema.

```
-- CreateTable
CREATE TABLE `Usuario` (
  `id` INTEGER NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(255) NULL,
  `email` VARCHAR(255) NOT NULL,
  `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
  `updatedAt` DATETIME(3) NOT NULL,
  UNIQUE INDEX `Usuario_email_key` (`email`),
  PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

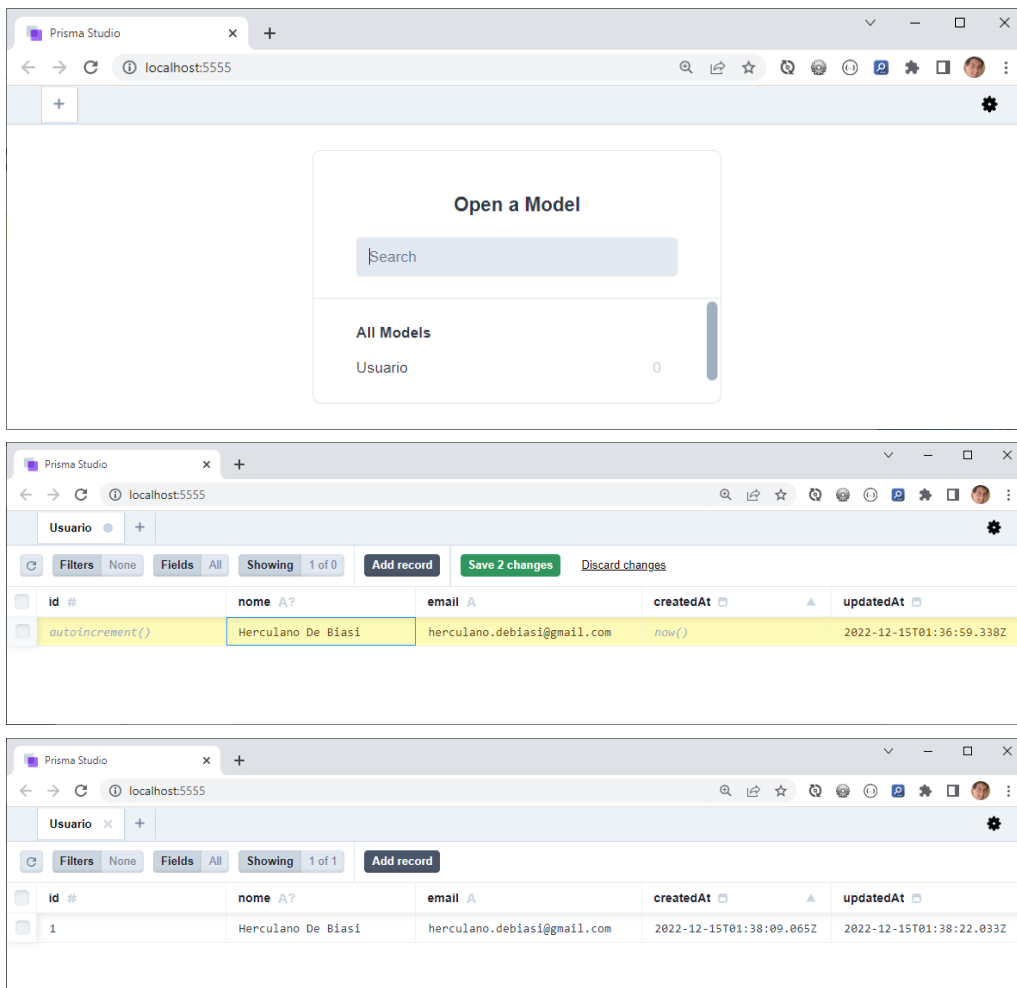
Pode-se ver também o banco de dados e tabela criados no MySQL Workbench.



The screenshot shows the MySQL Workbench interface. The left sidebar displays the database structure, including the `crud-node` database and the `usuario` table. The main window shows the details of the `usuario` table, including its columns, indexes, and privileges.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
id	int		NO	utf8mb4	utf8mb4_unicode...	select,insert,update,references	auto_increment
nome	varchar(255)		YES	utf8mb4	utf8mb4_unicode...	select,insert,update,references	
email	varchar(255)		NO	utf8mb4	utf8mb4_unicode...	select,insert,update,references	
createdAt	datetime(3)	CURRENT_TIMESTA...	NO			select,insert,update,references	DEFAULT_GENE...
updatedAt	datetime(3)		NO			select,insert,update,references	

5. Insira um registro na tabela usando o MySQL Workbench ou então usando a ferramenta descrita a seguir. Para inserir um registro através do MySQL Workbench, no campo `updatedAt` informe uma data como '2022-12-17'.
6. Prisma inclui uma ferramenta GUI que também permite visualizar e realizar operações sobre os dados.
 - `npx prisma studio`



7. Código do arquivo `server.js`.

```
lista17 - server.js
1  const express = require('express');
2  const cors = require('cors');
3  const formData = require('express-form-data');
4
5  const app = express();
6  const porta = 8081;
7
8  // Configura o CORS
9  app.use(cors({ origin: '*' }));
10
11 // Configura o tratamento das requisições POST
12 app.use(express.urlencoded({extended: true}));
13 app.use(express.json());
14
15 // Faz o parsing no formato multipart
16 app.use(formData.parse());
17
18 // Carrega o módulo rotas.js e define que as
19 // rotas iniciarão com o endereço '/api/usuarios'
20 app.use('/api/usuarios', require('./rotas'));
21
22 // Responde a requisição no endereço http://Localhost:8081/
23 app.get('/', (request, response) => {
24   response.status(200).send('Página home');
25 });
26
27 // Instancia o servidor
28 app.listen(porta,
29   () => console.log(`Servidor iniciado na porta: ${porta}`)
30 );
```

8. Crie o arquivo *rotas.js* dentro da pasta *src* com o código abaixo.

```
lista17 - rotas.js

1  const roteador = require('express').Router();
2  const { PrismaClient } = require("@prisma/client");
3
4  const { Usuario } = new PrismaClient();
5
6  roteador.get('/', async (req, res) => {
7    try {
8      const usuarios = await Usuario.findMany({
9        // select: {
10         //   id: true,
11         //   nome: true,
12         //   email: true
13        // },
14        // where: {
15         //   id: 1
16        // }
17      });
18
19      if (usuarios.length === 0) {
20        return res.status(200).json({mensagem: 'Não existem usuários cadastrados!'});
21      }
22
23      res.status(200).json({usuarios});
24    } catch (error) {
25      res.json(error);
26    }
27  });
28
29  module.exports = roteador;
```

9. Execute o servidor com o comando (certifique-se que você está na pasta *back-end-prisma*):

- `npm run devP`

10. Testando com o Postman. Crie uma nova requisição do tipo GET com a URL abaixo.

The image displays two screenshots of the Postman web interface, showing the results of API requests.

Top Screenshot: A GET request to `http://localhost:8081/api/usuarios` was successful, returning a 200 OK status. The response body, shown in JSON format, is:

```
{
  "usuarios": [
    {
      "id": 1,
      "nome": "Herculano De Biasi",
      "email": "herculano.debiasi@gmail.com",
      "createdAt": "2022-12-17T03:05:03.799Z",
      "updatedAt": "2022-12-17T03:05:03.799Z"
    }
  ]
}
```

Bottom Screenshot: The same GET request to `http://localhost:8081/api/usuarios` was successful, returning a 200 OK status. The response body, shown in JSON format, is:

```
{
  "mensagem": "Não existem usuários cadastrados!"
}
```

11. Inserindo usuários com a requisição POST.

```
JS rotas.js src JS ControllerUsuario.js controllers
back-end-prisma > src > JS rotas.js > ...
1  const roteador = require('express').Router();
2  const { PrismaClient } = require("@prisma/client");
3
4  const { Usuario } = new PrismaClient();
5
6  > roteador.get('/', async (req, res) => { ...
27  });
28
29  roteador.post('/', async (req, res) => {
30    const { nome, email } = req.body;
31
32    try {
33      let usuario = await Usuario.findUnique({ where: { email } });
34      if (usuario) {
35        return res.status(401).json({ error: 'E-mail já cadastrado!' });
36      }
37
38      usuario = await Usuario.create({
39        data: {
40          nome,
41          email
42        }
43      });
44
45      res.status(200).json({ mensagem: 'Usuário incluído com sucesso!', usuario });
46    } catch (error) {
47      res.json(error);
48    }
49  });
50
51  module.exports = roteador;
```

Testando com o Postman. Crie uma nova requisição do tipo POST chamada 'Criar usuário' com a URL abaixo. Configure para Body → Raw → JSON.

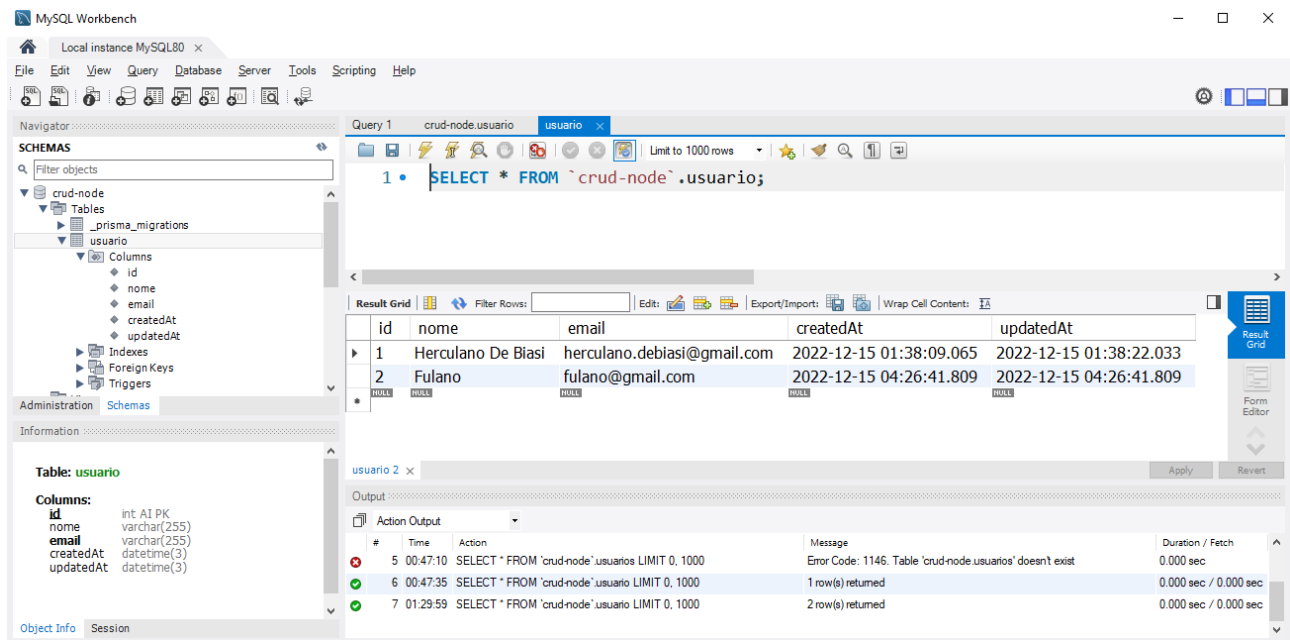
The screenshot shows the Postman interface with a new POST request named 'Criar usuário' at the URL 'http://localhost:8081/api/usuarios'. The request body is raw JSON:

```
{  "nome": "Herculano De Biasi",  "email": "herculano.debiasi@gmail.com"}
```

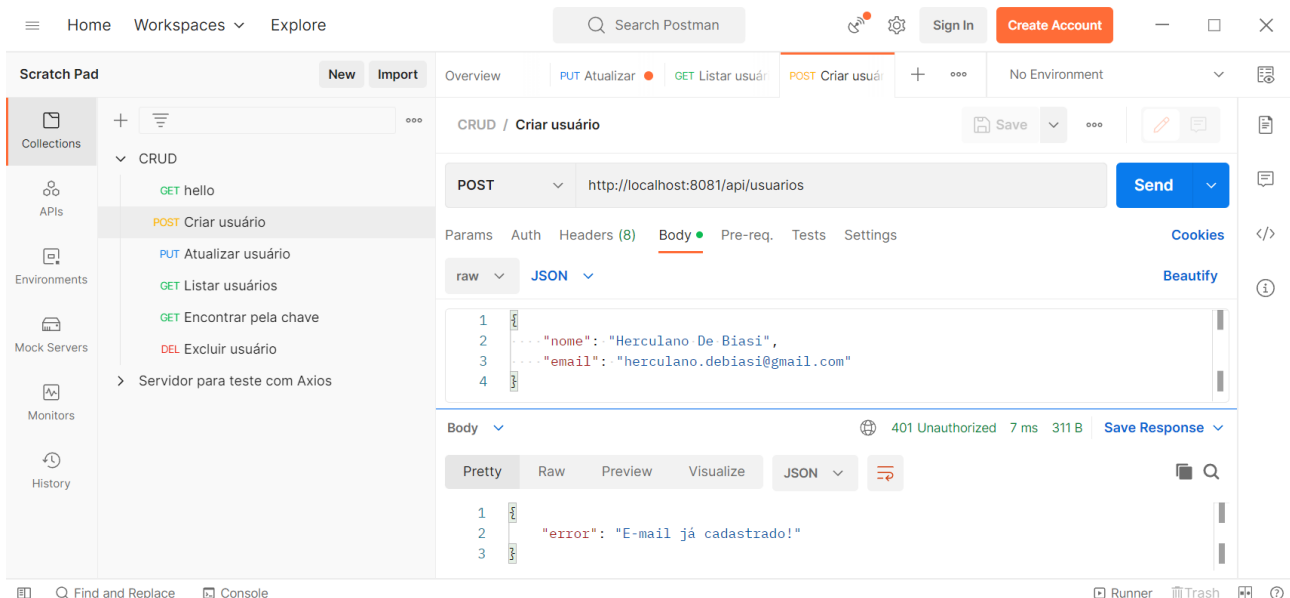
. The response is shown in the 'Body' tab, displaying a success message and the created user object:

```
{  "mensagem": "Usuário incluindo com sucesso",  "usuario": {    "id": 1,    "nome": "Herculano De Biasi",    "email": "herculano.debiasi@gmail.com",    "createdAt": "2022-12-16T15:16:06.008Z",    "updatedAt": "2022-12-16T15:16:06.008Z"  }}
```

Verificando no Workbench.



Tentando inserir um registro com e-mail já existente.



12. A seguir o código será refatorado por motivos de organização. Crie uma pasta chamada *controllers* dentro de *src* e dentro dela um arquivo chamado *controladorUsuario.js*.

O código refatorado do arquivo *rotas.js* é mostrado abaixo:

```
lista17 - rotas.js

1 const controlador = require('./controllers/controladorUsuario');
2
3 const roteador = require('express').Router();
4
5 roteador.get('/', controlador.listarUsuarios);
6 roteador.post('/', controlador.criarUsuario);
7
8 module.exports = roteador;
```


O código refatorado do arquivo *controladorUsuario.js* é mostrado abaixo:

```
lista17 - controladorUsuario.js

1  const { PrismaClient } = require("@prisma/client");
2  const { Usuario } = new PrismaClient();
3
4  module.exports = {
5    async listarUsuarios(req, res) {
6      try {
7        const usuarios = await Usuario.findMany({
8          // select: {
9            //   id: true,
10           //   nome: true,
11           //   email: true
12         // },
13         // where: {
14           //   id: 1
15         // }
16       });
17
18       if (usuarios.length === 0) {
19         return res.status(200).json({mensagem: 'Não existem usuários cadastrados!'});
20       }
21
22       res.status(200).json({usuarios});
23     } catch (error) {
24       res.json(error);
25     }
26   },
27
28   async criarUsuario(req, res) {
29     const { nome, email } = req.body;
30     // console.log(nome, email);
31
32     try {
33       let usuario = await Usuario.findUnique({ where: { email } });
34       if (usuario) {
35         return res.status(401).json({ error: 'E-mail já cadastrado!'});
36       }
37
38       usuario = await Usuario.create({
39         data: {
40           nome,
41           email
42         }
43       });
44
45       res.status(200).json({ mensagem: 'Usuário incluído com sucesso!', usuario });
46     } catch (error) {
47       res.json(error);
48     }
49   }
50 }
```

13. Crie uma nova rota (linha 7) que irá buscar um registro específico por meio de sua chave primária.

```
lista17 - rotas.js

1  const controlador = require('./controllers/controladorUsuario');
2
3  const roteador = require('express').Router();
4
5  roteador.get('/', controlador.listarUsuarios);
6  roteador.post('/', controlador.criarUsuario);
7  roteador.get('/:id', controlador.buscarUsuarioPorID);
8
9  module.exports = roteador;
```

O código novo no arquivo *controladorUsuario.js*. Não esqueça de acrescentar a vírgula ao final da linha 48:

```
JS controladorUsuario.js controllers X
back-end-prisma > src > controllers > JS controladorUsuario.js > ...
1  const { PrismaClient } = require("@prisma/client");
2  const { Usuario } = new PrismaClient();
3
4  module.exports = {
5  >   async listarUsuarios(req, res) { ...
26  },
27
28  >   async criarUsuario(req, res) { ...
48  },
49
50   async buscarUsuarioPorID(req, res) {
51     try {
52       const { id } = req.params;
53
54       const usuario = await Usuario.findUnique({ where: { id: Number(id) } });
55       if (!usuario) {
56         return res.status(200).json({ mensagem: 'Usuário não encontrado!' });
57       }
58
59       res.status(200).json(usuario);
60     } catch (error) {
61       res.json(error);
62     }
63   }
64 }
```

Testando com o Postman. Crie uma nova requisição do tipo GET com a URL abaixo.

The screenshot shows the Postman interface with a GET request to `http://localhost:8081/api/usuarios/1`. The response is a JSON object with the following data:

```
{
  "id": 1,
  "nome": "Herculano De Biasi",
  "email": "herculano.debiasi@gmail.com",
  "createdAt": "2022-12-15T05:08:58.284Z",
  "updatedAt": "2022-12-15T05:08:58.284Z"
}
```

Caso não seja encontrado um registro com a chave informada, o Postman retornará o seguinte.

The screenshot shows the Postman interface with a GET request to `http://localhost:8081/api/usuarios/2`. The response is a JSON object with the following data:

```
{
  "mensagem": "Usuário não encontrado!"
}
```


14. Para alterar um usuário crie a rota abaixo (linha 8):

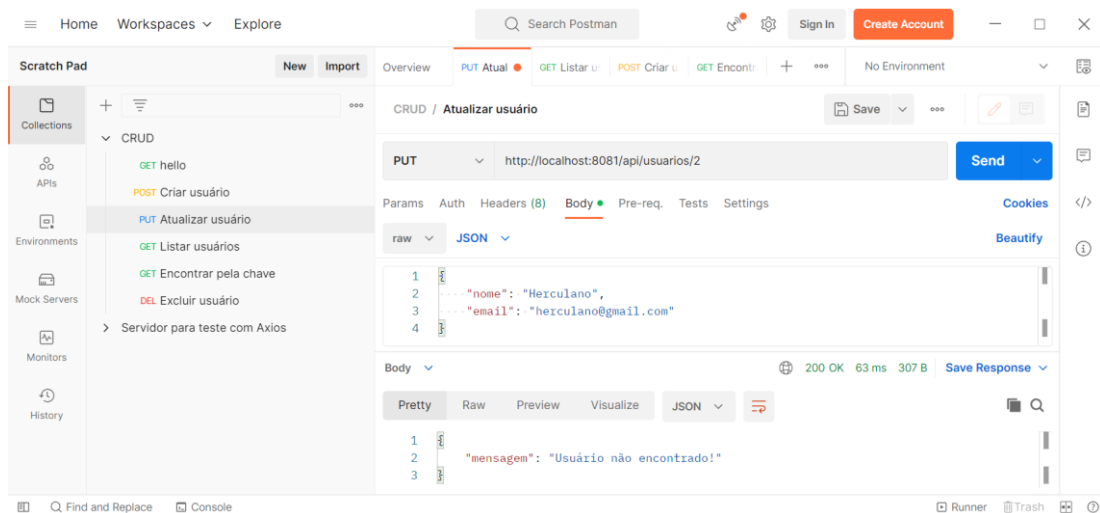
```
lista17 - rotas.js

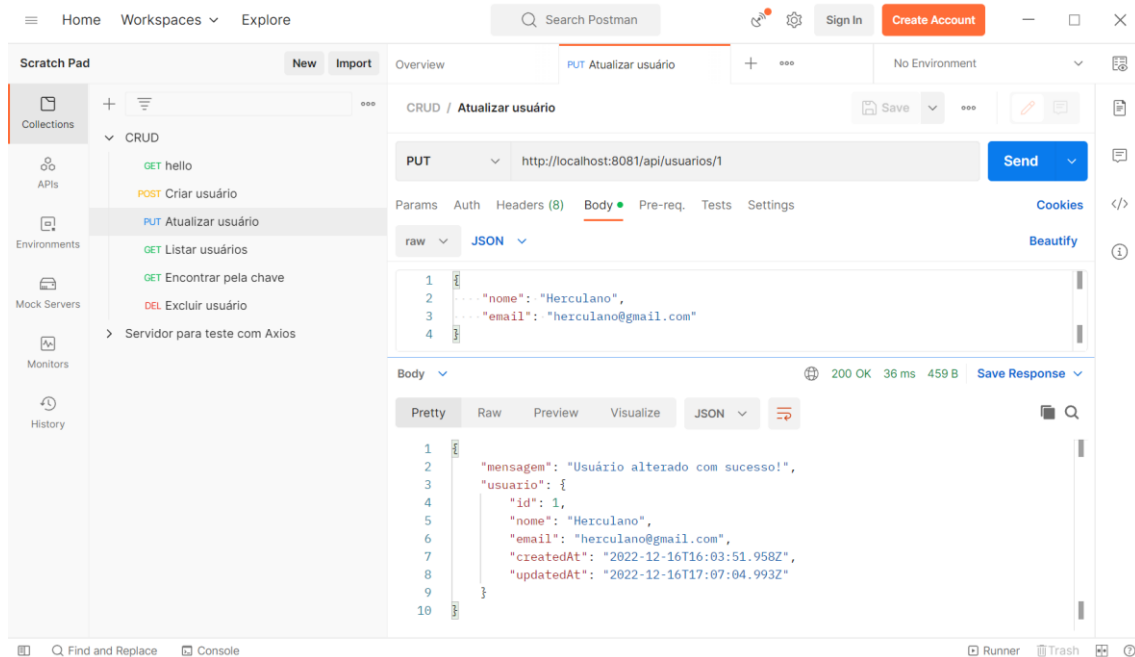
1  const controlador = require('./controllers/controladorUsuario');
2
3  const roteador = require('express').Router();
4
5  roteador.get('/', controlador.listarUsuarios);
6  roteador.post('/', controlador.criarUsuario);
7  roteador.get('/:id', controlador.buscarUsuarioPorID);
8  roteador.put('/:id', controlador.alterarUsuario);
9
10 module.exports = roteador;
```

O código no arquivo *controladorUsuario.js*. Não esqueça de acrescentar a vírgula ao final da linha 64:

```
JS controladorUsuario.js controllers X
back-end-prisma > src > controllers > JS controladorUsuario.js > ...
1  const { PrismaClient } = require("@prisma/client");
2  const { Usuario } = new PrismaClient();
3
4  module.exports = {
5  >   async listarUsuarios(req, res) { ...
26   },
27
28   >   async criarUsuario(req, res) { ...
49   },
50
51   >   async buscarUsuarioPorID(req, res) { ...
64   },
65
66   async alterarUsuario(req, res) {
67     try {
68       const { id } = req.params;
69       const { nome, email } = req.body;
70
71       let usuario = await Usuario.findUnique({ where: { id: Number(id) } });
72       if (!usuario) {
73         return res.status(200).json({ mensagem: 'Usuário não encontrado!' });
74       }
75
76       usuario = await Usuario.update({
77         where: { id: Number(id) },
78         data: { nome, email }
79       });
80
81       res.status(200).json({ mensagem: 'Usuário alterado com sucesso!', usuario });
82     } catch (error) {
83       res.json(error);
84     }
85   }
86 }
```

Testes com o Postman:





15. Para remover um usuário crie o método abaixo. Não esqueça de acrescentar a vírgula ao final da linha 85:

```
JS controladorUsuario.js controllers X
back-end-prisma > src > controllers > JS controladorUsuario.js > ...
1  const { PrismaClient } = require("@prisma/client");
2  const { Usuario } = new PrismaClient();
3
4  module.exports = {
5  >   async listarUsuarios(req, res) {...
26  },
27
28  >   async criarUsuario(req, res) {...
49  },
50
51  >   async buscarUsuarioPorID(req, res) {...
64  },
65
66  >   async alterarUsuario(req, res) {...
85  },
86
87   async excluirUsuario(req, res) {
88     try {
89       const { id } = req.params;
90
91       const usuario = await Usuario.findUnique({ where: { id: Number(id) } });
92       if (!usuario) {
93         return res.status(200).json({mensagem: 'Usuário não encontrado!'});
94       }
95
96       await Usuario.delete({ where: { id: Number(id) } });
97       res.status(200).json({ mensagem: 'Usuário excluído!' });
98     } catch (error) {
99       res.json(error);
100     }
101   }
102 }
```

Crie a nova rota (DELETE) no arquivo *rotas.js* (linha 9).

```
lista17 - rotas.js

1  const controlador = require('./controllers/controladorUsuario');
2
3  const roteador = require('express').Router();
4
5  roteador.get('/', controlador.listarUsuarios);
6  roteador.post('/', controlador.criarUsuario);
7  roteador.get('/:id', controlador.buscarUsuarioPorID);
8  roteador.put('/:id', controlador.alterarUsuario);
9  roteador.delete('/:id', controlador.excluirUsuario);
10
11 module.exports = roteador;
```

Testando com o Postman. Crie uma nova requisição do tipo DELETE com a URL abaixo.

The screenshot shows the Postman interface with a DELETE request configured. The URL is `http://localhost:8081/api/usuarios/1`. The response is a 200 OK status with a response time of 22 ms and a body of `{\"mensagem\": \"Usuário excluído!\"}`.

Caso não seja encontrado um registro com a chave informada, o Postman retornará o seguinte.

The screenshot shows the Postman interface with a DELETE request configured. The URL is `http://localhost:8081/api/usuarios/1`. The response is a 200 OK status with a response time of 16 ms and a body of `{\"mensagem\": \"Usuário não encontrado!\"}`.