ALTERAÇÕES NO PROJETO

Instruções

- Executar o comando **npm install** para instalar os pacotes necessários.
- Executar o comando **npm run dev** para executar o servidor**.**
- Adicionar o método getByEmail na interface IUserRepository que se encontra no arquivo userRepository.interface.ts.

```
export interface IUserRepository extends IDefaultRepository<IUserEntity, IUserPayload> {
    getByEmail: (email: string) => Promise<IUserEntity | null>
}
```

- Adicionar o método getByEmail na classe UserRepository que se encontra no arquivo userRepository.repository.ts.

```
31
32    async getByEmail(email: string): Promise<IUserEntity | null> {
33      try {
34        const user = await prismaClient.user.findUnique({
35          where: {
36            email,
37          },
38        });
39
40        return user;
41      } catch (error: any) {
42        throw new Error(error);
43      }
44    }
```

- Adicionar o método getByEmail na interface IUserService que se encontra no arquivo userService.interface.ts.

```
4
5   export interface IUserService<Payload, ServiceReturn>
6     extends IDefaultService<Payload, ServiceReturn> {
7     getByEmail: (email: string) => Promise<IServiceReturn>;
8   }
9
```

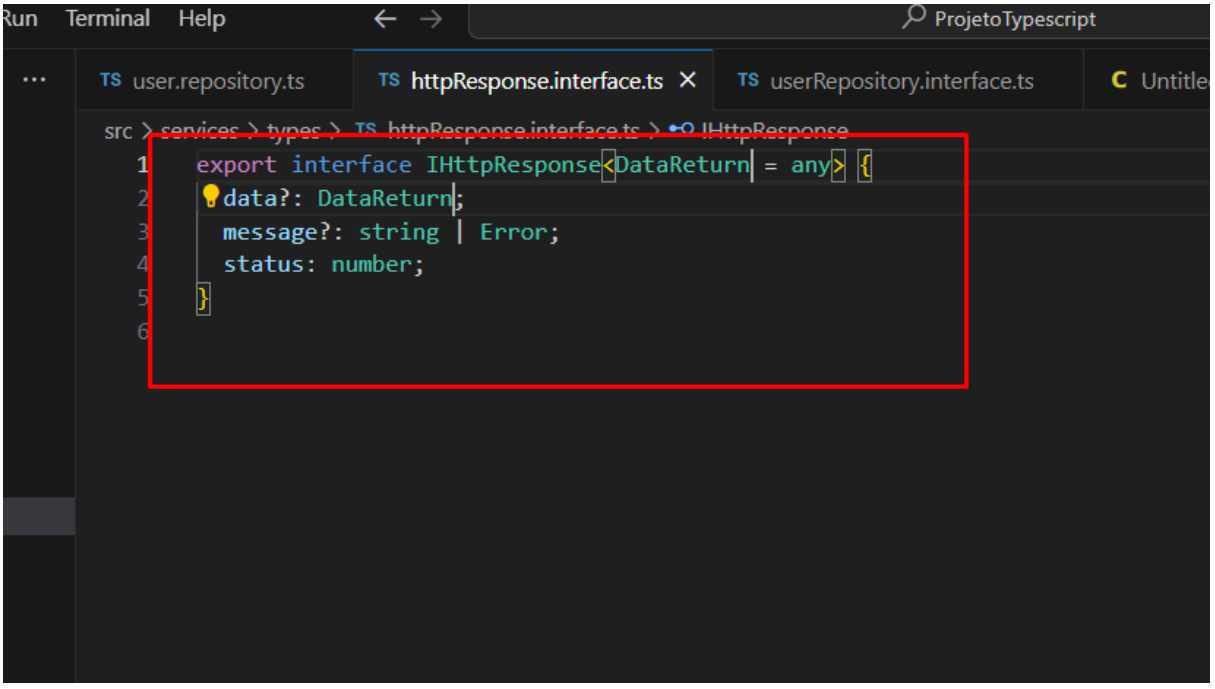
- Adicionar o método getByEmail na classe UserService que se encontra no arquivo user.service.ts.

```
async getByEmail(email: string): Promise<IServiceReturn> {
  try {
    const user = await this.userRepository.getByEmail(email);

    if (!user) {
      return {
        status: 404,
        message: "Dado não encontrado.",
      };
    }

    return {
      status: 200,
      data: user,
    };
  } catch (error: any) {
    return {
      status: 500,
      message: "Ops! Algo aconteceu internamente.",
    };
  }
}
```

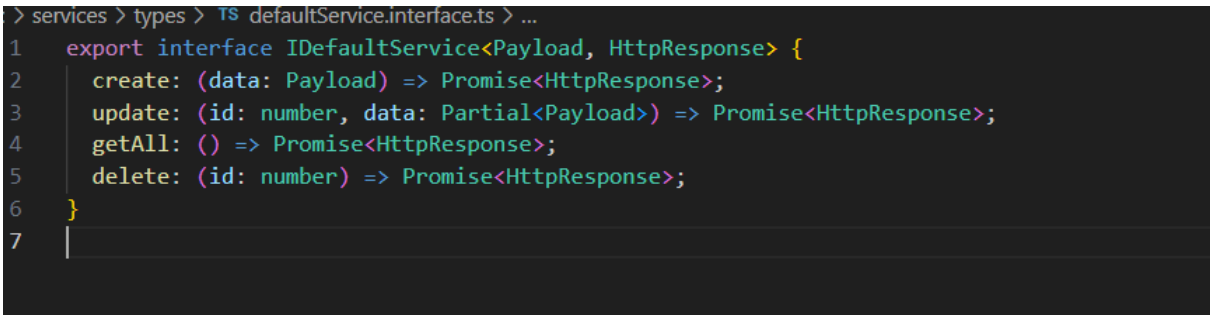
- Adicionar validação no arquivo user.route.ts para lidar com listagem de usuário por email.

```
userRouter.get("/", async (request, response) => {
  const { email } = request.query;

  if (email) {
    const { status, ...returnData } = await userService.getByEmail(
      email as string
    );

    return response.status(status).json(returnData);
  }

  const { status, ...returnData } = await userService.getAll();

  return response.status(status).json(returnData);
});
```

- Criar a interface IHttpResponse dentro de um arquivo chamado httpResponse.interface.ts.



- Remover os anys dos retornos da interface IDefaultService e colocar o tipo genérico HttpResponse.



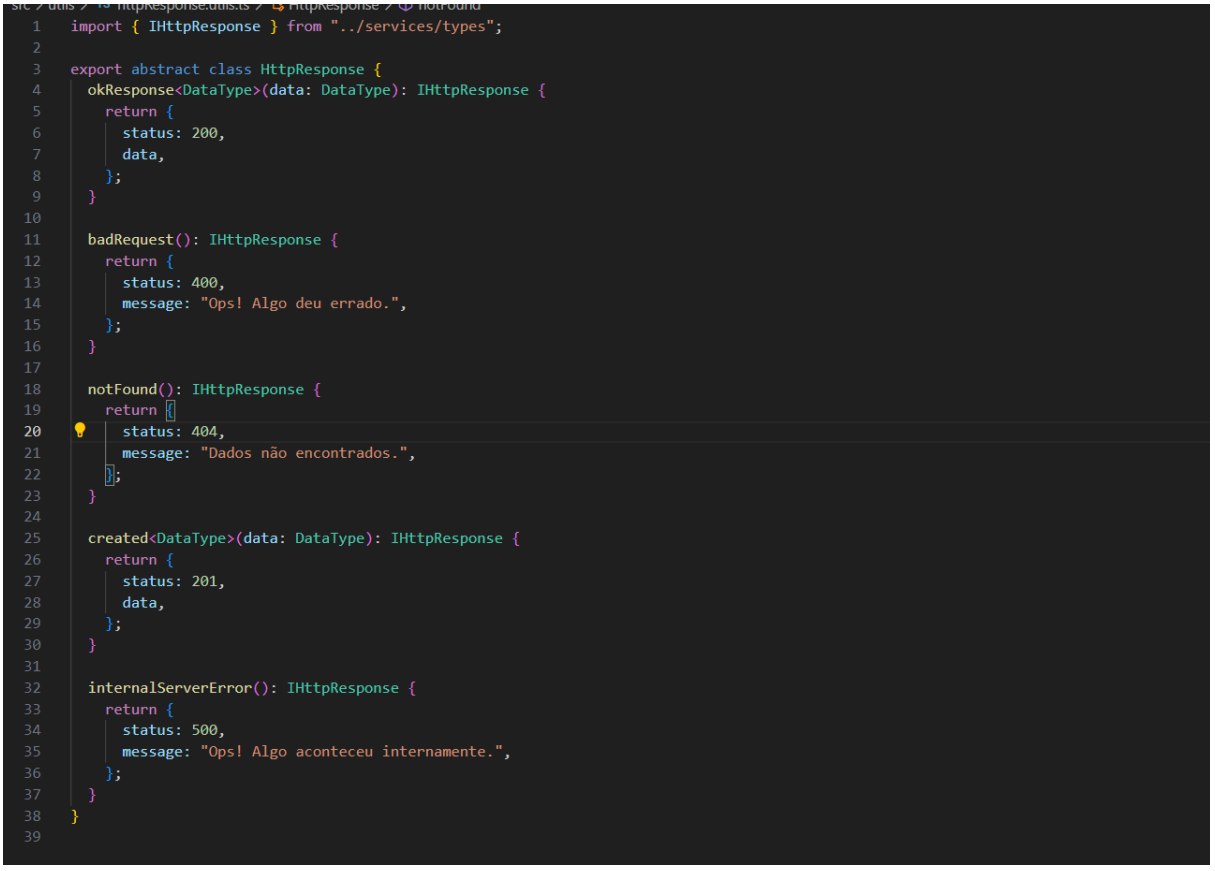
- Adicionar o tipo genérico HttpResponse na interface IUserService.
- Colocar o tipo HttpResponse como retorno do método getByEmail.

```
export interface IUserService<Payload, HttpResponse>
  extends IDefaultService<Payload, HttpResponse> {}
```

- Na classe UserService, informar a interface IHttpResponse como parâmetro para a interface IUserService e nas promises tambem.

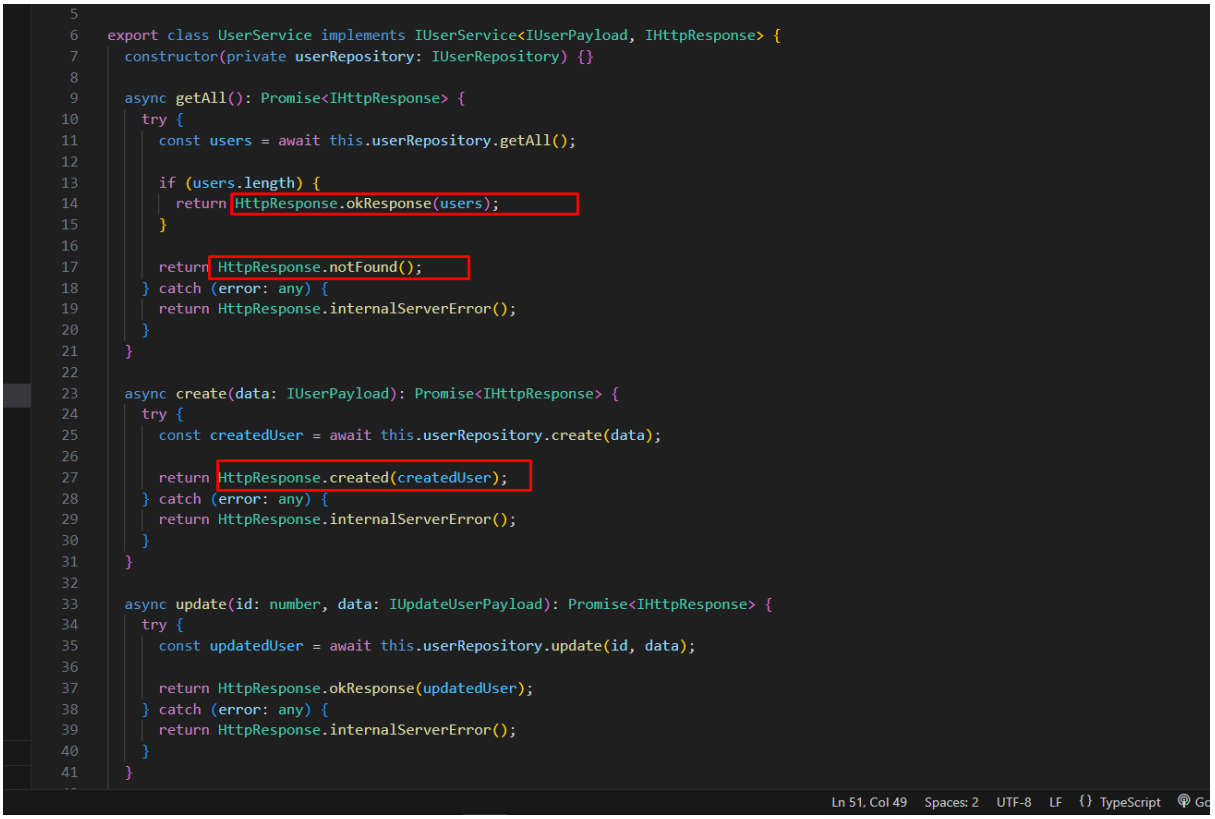
```
export class UserService implements IUserService<IUserPayload, IHttpResponse> {
  constructor(private userRepository: IUserRepository) {}

  async getAll(): Promise<IHttpResponse> {
    try {
      const users = await this.userRepository.getAll();

      if (users.length) {
        return {
          status: 200,
          data: users,
        };
      }

      return {
        status: 404,
        message: "Dados não encontrados.",
      };
    } catch (error: any) {
      return {
        status: 500,
        message: "Ops! Algo aconteceu internamente.",
      };
    }
  }

  async create(data: IUserPayload): Promise<IHttpResponse> {
    try {
      const createdUser = await this.userRepository.create(data);

      return {
        status: 201,
        data: createdUser,
      };
    } catch (error: any) {
      return {
        status: 500,
```

- Criar uma pasta chamada **utils** e dentro dela criar um arquivo chamado httpResponse.utils.ts e dentro desse arquivo criar a classe HttpResponse.



```ts
import { IHttpResponse } from "../services/types";

export abstract class HttpResponse {
  okResponse<DataType>(data: DataType): IHttpResponse {
    return {
      status: 200,
      data,
    };
  }

  badRequest(): IHttpResponse {
    return {
      status: 400,
      message: "Ops! Algo deu errado.",
    };
  }

  notFound(): IHttpResponse {
    return {
      status: 404,
      message: "Dados não encontrados.",
    };
  }

  created<DataType>(data: DataType): IHttpResponse {
    return {
      status: 201,
      data,
    };
  }

  internalServerError(): IHttpResponse {
    return {
      status: 500,
      message: "Ops! Algo aconteceu internamente.",
    };
  }
}
```

- Utilizar a classe HttpResponse no arquivo user.service.ts em todos os lugares onde retornamos um objeto contendo um campo status e o campo data ou message.
- Utilização dos métodos:
  - Utilizar *return HttpResponse.okResponse()* onde o status for 200;
  - Utilizar *return HttpResponse.createdResponse()* onde o status for 201;
  - Utilizar *return HttpResponse.notFoundResponse()* onde o status for 404;
  - Utilizar *return HttpResponse.badRequestResponse()* onde o status for 400;
  - Utilizar *return HttpResponse.internalServerErrorResponse()* onde o status for 500;



```
export class UserService implements IUserService<IUserPayload, IHttpResponse> {
  constructor(private userRepository: IUserRepository) {}

  async getAll(): Promise<IHttpResponse> {
    try {
      const users = await this.userRepository.getAll();

      if (users.length) {
        return HttpResponse.okResponse(users);
      }

      return HttpResponse.notFound();
    } catch (error: any) {
      return HttpResponse.internalServerError();
    }
  }

  async create(data: IUserPayload): Promise<IHttpResponse> {
    try {
      const createdUser = await this.userRepository.create(data);

      return HttpResponse.created(createdUser);
    } catch (error: any) {
      return HttpResponse.internalServerError();
    }
  }

  async update(id: number, data: IUpdateUserPayload): Promise<IHttpResponse> {
    try {
      const updatedUser = await this.userRepository.update(id, data);

      return HttpResponse.okResponse(updatedUser);
    } catch (error: any) {
      return HttpResponse.internalServerError();
    }
  }
```

Ln 51, Col 49 Spaces: 2 UTF-8 LF {} TypeScript Go