

Sumário

Repositório aplicação inicial	1
Criar uma instancia do mongo no Docker	1
Configurar a conexão no app.....	2
Criar a Model	2
Criar a repository	3
Criar a Controller	4

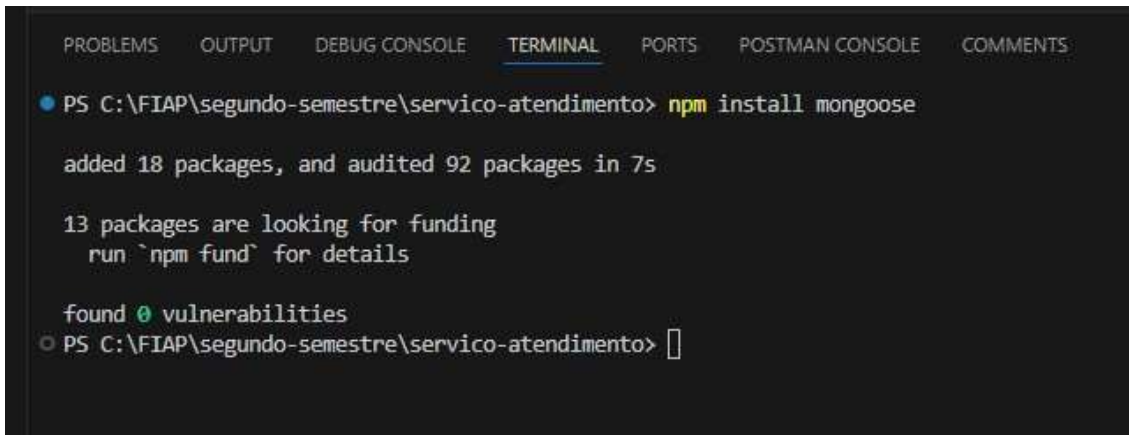
Repositório aplicação inicial

<https://github.com/professorthiagoxavier/microservice/branches>

branch: aula-1-status-code

Instalar o mongoose

1 - Abrir o cmd e utilizar o comando:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

● PS C:\FIAP\segundo-semester\servico-atendimento> npm install mongoose

added 18 packages, and audited 92 packages in 7s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
○ PS C:\FIAP\segundo-semester\servico-atendimento> 
```

Criar uma instancia do mongo no Docker

É importante configurar se o Docker está iniciado.

Após isso, vamos utilizar o Docker-compose disponibilizado no Repositório para iniciar a instância.

`docker-compose up -d`

```
bash
```

```
docker-compose up -d
```

Se tiver algum problema do container subir e cair em seguida é problema no compose

Pode dar um Docker-compose down pra excluir tudo e depois o up -d denovo quando consertar

Pode dar Docker logs “nome do container”

Pode dar docker ps -a para ver todas imagens disponíveis

`docker exec -it “nome do container”/bin/bash`

`mongosh --username root --password example --authenticationDatabase admin` (nesse caso o docker-compose estava configurado para root e example)

Docker-compose do professor estava errado, não tava rodando o mongo-express, o do “hub.docker.com” funcionou

https://hub.docker.com/_/mongo

Testando mongo:

`show dbs` (admin, config, local = indica que nenhum banco de dados adicional foi criado)

`use “nome de um banco”`

`show collections`

Criar uma coleção: `db.createCollection("products")`

Inserir um documento: `db.products.insert({ name: "Produto 1", price: 100 })`

Verificar se o documento foi inserido: `db.products.find()`

Dá npm start pra testar no postman as outras coisas, get/post dos produtos

No fim do projeto você deve pausar/parar os containers:

1. Pausar os contêineres (docker pause): se você vai continuar trabalhando no projeto mais tarde e quer manter os contêineres "prontos", mas não quer que eles estejam consumindo recursos (CPU/memória). Isso preserva o estado do contêiner.

`docker pause nome_ou_id_do_container`

`docker unpause nome_ou_id_do_container`

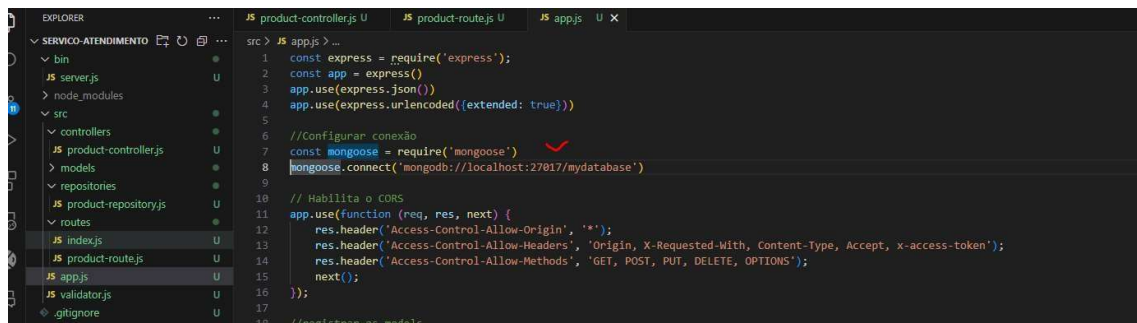
2. Parar os contêineres (docker stop): se você não precisa que os contêineres fiquem em execução agora, mas deseja manter o contêiner (e seus volumes) para retomá-lo facilmente depois.

`docker stop nome_ou_id_do_container`

`docker start nome_ou_id_do_container`

Configurar a conexão no app

Abra o arquivo app.js e configure a conexão;



```
src > JS appjs > ...
1  const express = require('express');
2  const app = express();
3  app.use(express.json());
4  app.use(express.urlencoded({extended: true}));
5
6  //Configurar conexão
7  const mongoose = require('mongoose');
8  mongoose.connect('mongodb://localhost:27017/mydatabase');
9
10 // Habilita o CORS
11 app.use(function (req, res, next) {
12   res.header('Access-Control-Allow-Origin', '*');
13   res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, x-access-token');
14   res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
15   next();
16 });
17
18 //registrar os models
```

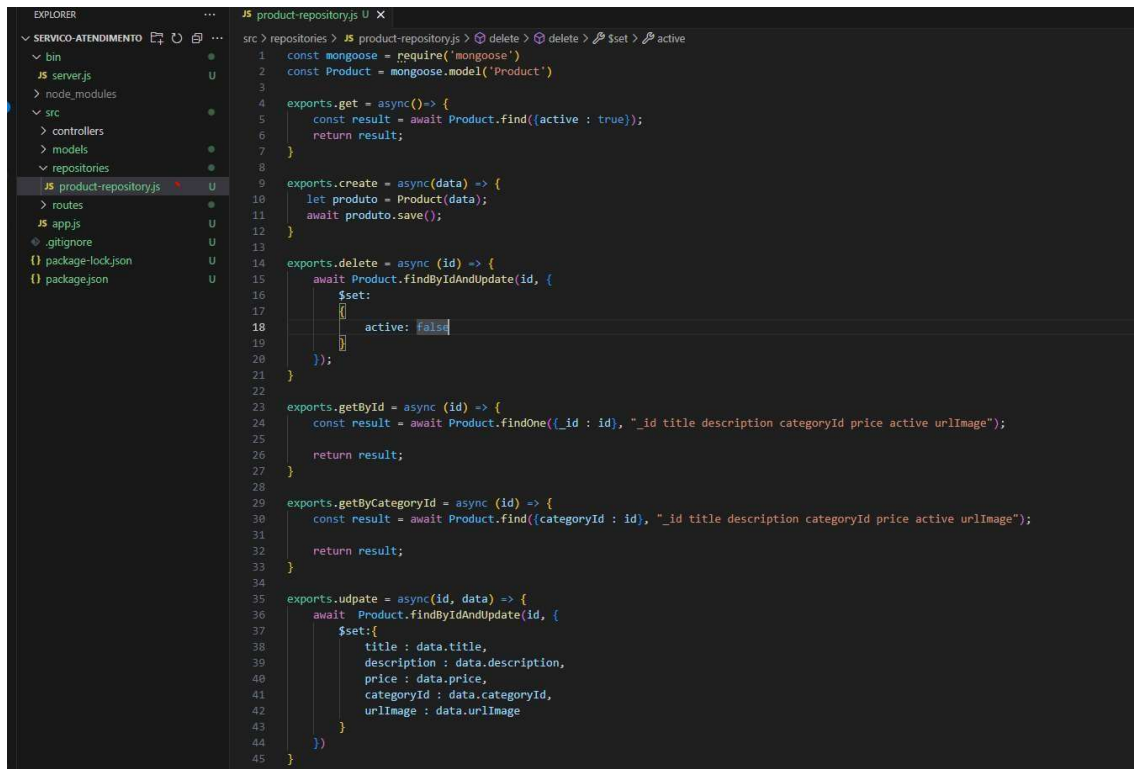
Criar a Model

1. Crie a pasta na seguinte estrutura src/models
2. Crie um arquivo chamado produto.js

```
src > models > JS product.js > ...
1 | 'use strict'
2 const mongoose = require('mongoose')
3 const Schema = mongoose.Schema;
4
5 const schema = new Schema({
6   title: {
7     type: String,
8     required: true,
9     trim: true
10  },
11  description: {
12    type: String,
13    required: true
14  },
15  categoryId: {
16    type: String
17  },
18  urlImage: {
19    type: String
20  },
21  price: {
22    type: Number,
23    required: true
24  },
25  active: {
26    type: Boolean,
27    required: true,
28    default: true
29  }
30 });
31
32 module.exports = mongoose.model('Product', schema)
33
```

Criar a repository

1. Crie a pasta na seguinte estrutura src/repository
2. Crie um arquivo chamado produto-repository.js

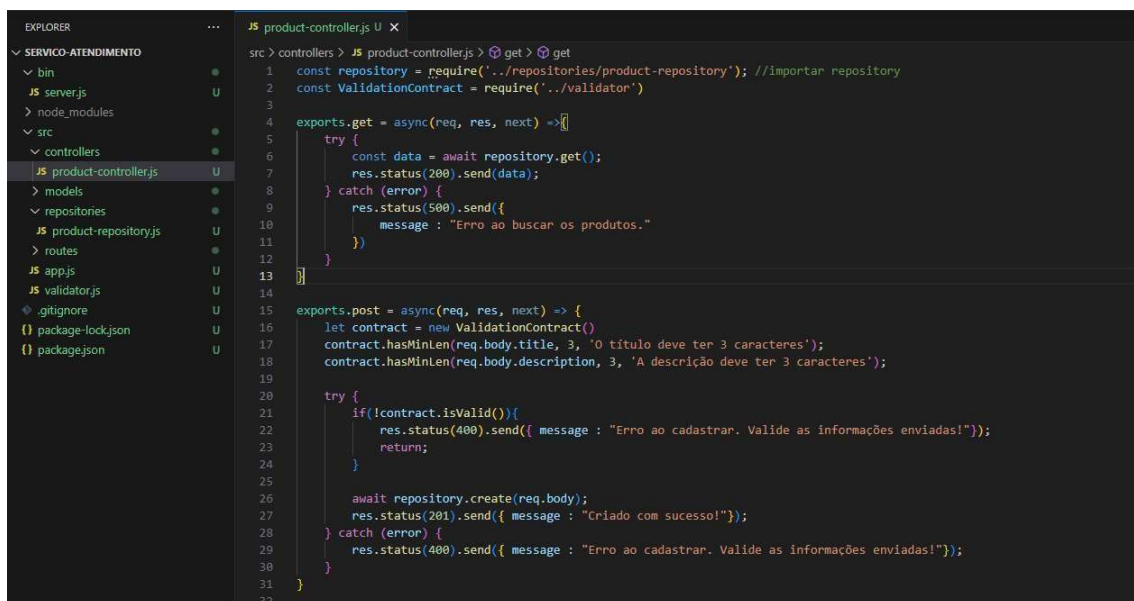


```
1 const mongoose = require('mongoose')
2 const Product = mongoose.model('Product')
3
4 exports.get = async () => {
5   const result = await Product.find({active : true});
6   return result;
7 }
8
9 exports.create = async(data) => {
10   let produto = Product(data);
11   await produto.save();
12 }
13
14 exports.delete = async (id) => {
15   await Product.findByIdAndUpdate(id, {
16     $set: {
17       active: false
18     }
19   });
20 }
21
22
23 exports.getById = async (id) => {
24   const result = await Product.findOne({_id : id}, "_id title description categoryId price active urlImage");
25   return result;
26 }
27
28
29 exports.getByCategoryId = async (id) => {
30   const result = await Product.find({categoryId : id}, "_id title description categoryId price active urlImage");
31   return result;
32 }
33
34
35 exports.udpate = async(id, data) => {
36   await Product.findByIdAndUpdate(id, {
37     $set:{
38       title : data.title,
39       description : data.description,
40       price : data.price,
41       categoryId : data.categoryId,
42       urlImage : data.urlImage
43     }
44   })
45 }
```

Criar a Controller

Ponto de atenção para o arquivo de validação, para tratarmos bad requests (400)

Get e post



```
1 const repository = require('../repositories/product-repository'); //importar repository
2 const ValidationContract = require('../validator')
3
4 exports.get = async(req, res, next) =>{
5   try {
6     const data = await repository.get();
7     res.status(200).send(data);
8   } catch (error) {
9     res.status(500).send({
10       message : "Erro ao buscar os produtos."
11     })
12   }
13 }
14
15 exports.post = async(req, res, next) => {
16   let contract = new ValidationContract()
17   contract.hasMinlen(req.body.title, 3, 'O título deve ter 3 caracteres');
18   contract.hasMinlen(req.body.description, 3, 'A descrição deve ter 3 caracteres');
19
20   try {
21     if(!contract.isValid()){
22       res.status(400).send({ message : "Erro ao cadastrar. Valide as informações enviadas!"});
23       return;
24     }
25
26     await repository.create(req.body);
27     res.status(201).send({ message : "Criado com sucesso!"});
28   } catch (error) {
29     res.status(400).send({ message : "Erro ao cadastrar. Valide as informações enviadas!"});
30   }
31 }
32 }
```

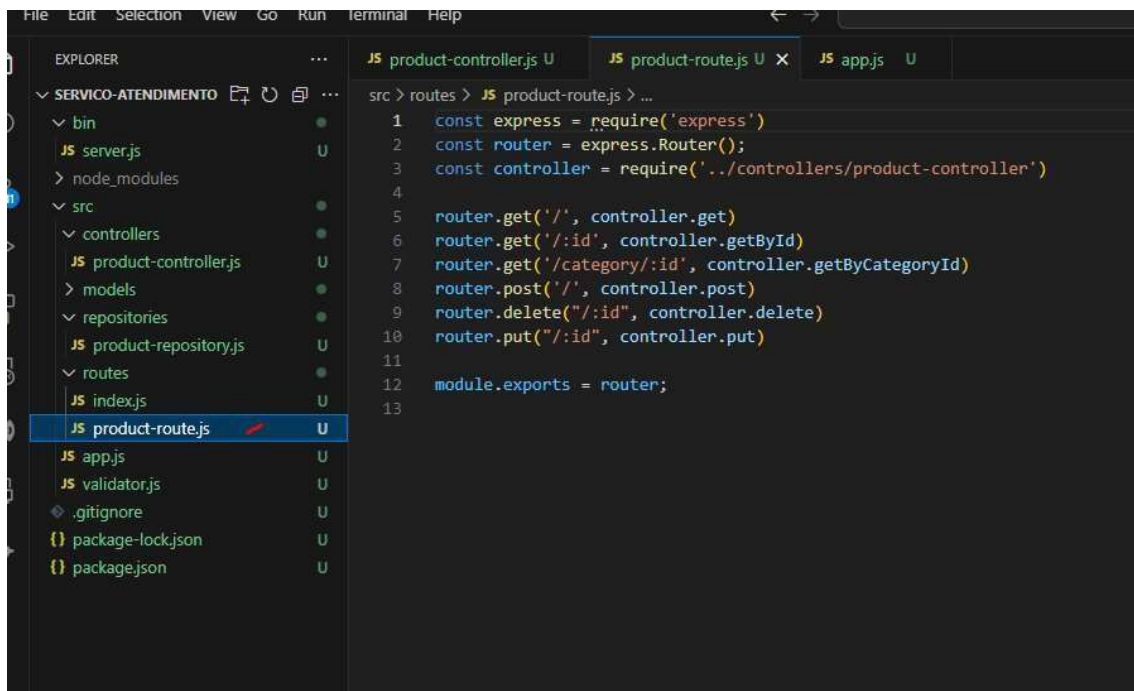
Put, GetById e Delete

```
33 exports.put = async(req, res, next) => {
34   let contract = new ValidationContract();
35   contract.hasMinLen(req.body.title, 3, 'O título deve ter 3 caracteres');
36   contract.hasMinLen(req.body.description, 3, 'A descrição deve ter 3 caracteres');
37
38   try {
39     if(!contract.isValid()){
40       res.status(400).send({ message : "Erro ao cadastrar. Valide as informações enviadas!"});
41       return;
42     }
43     const id = req.params.id;
44     const body = req.body;
45     await repository.update(id, body);
46     res.status(204).send({ message : "Atualizado!"});
47   } catch (error) {
48     res.status(400).send({ message : "Erro ao cadastrar. Valide as informações enviadas!"});
49   }
50 }
51
52
53 exports.getById = async(req, res, next) => {
54   try {
55     const data = await repository.getById(req.params.id);
56     if(data == null)
57       res.status(404).send();
58     res.status(200).send(data);
59   } catch (error) {
60     res.status(500).send({
61       message : "Erro ao buscar o produto por categoria."
62     });
63   }
64 }
65
66
67 exports.delete = async(req, res, next) => {
68   const id = req.params.id;
69   let response = {
70     id: id,
71     message: "Removed"
72   };
73   try {
74     await repository.delete(id);
75     res.status(204).send(response);
76   } catch (error) {
77     res.status(400).send({ message: 'Erro ao remover', error });
78   }
79 }
80 }
```

Salvar o arquivo validator.js (arquivo está no repositório na branch da aula) no mesmo nível do app.js.

Criar a rota Product

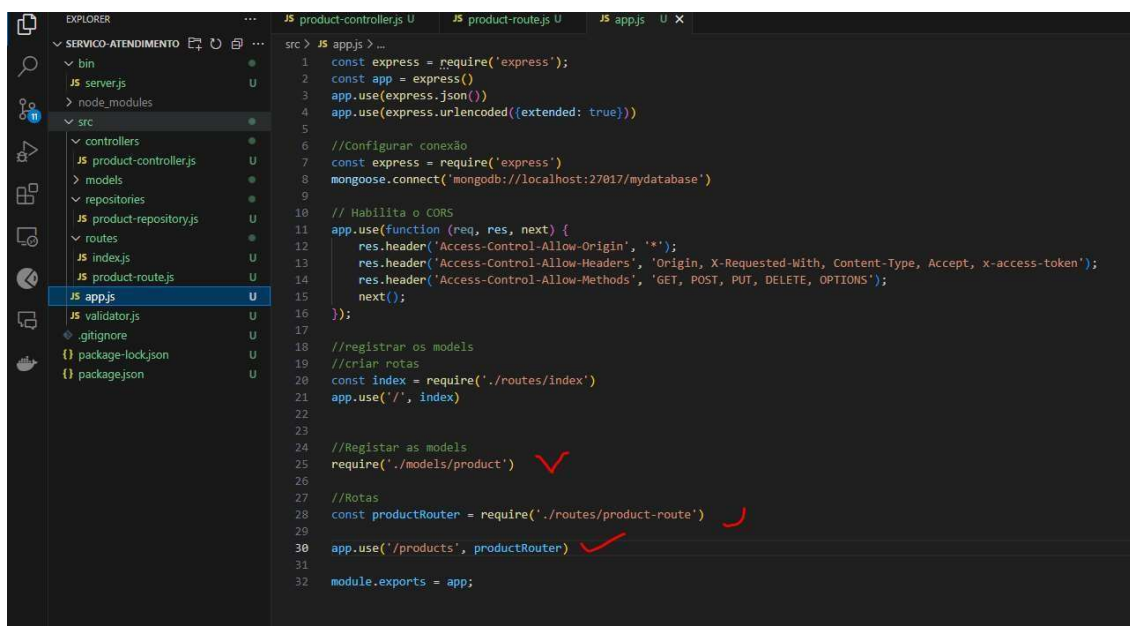
1. Abra a pasta de rota e adicione o arquivo product-route.js



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure for 'SERVICO-ATENDIMENTO'. The 'src' directory is expanded, showing subdirectories like 'controllers', 'models', 'repositories', and 'routes'. The 'product-route.js' file in the 'routes' directory is selected and highlighted. The main editor area shows the code for 'product-route.js', which defines a router with several routes: a GET route for '/', a GET route for '/:id', a GET route for '/category/:id', a POST route for '/', a DELETE route for '/:id', and a PUT route for '/:id'. The router is then exported as 'module.exports'.

```
src > routes > JS product-route.js > ...
1  const express = require('express')
2  const router = express.Router();
3  const controller = require('../controllers/product-controller')
4
5  router.get('/', controller.get)
6  router.get('/:id', controller.getById)
7  router.get('/category/:id', controller.getCategoryId)
8  router.post('/', controller.post)
9  router.delete('/:id', controller.delete)
10 router.put('/:id', controller.put)
11
12 module.exports = router;
13
```

Registrar a rota e a Model



The screenshot shows the VS Code interface with the 'app.js' file selected in the Explorer sidebar. The main editor area displays the code for 'app.js', which sets up the Express application. It includes the following code: initialization of Express, setting of view engines, connection to MongoDB, enabling of CORS, and registration of routes and models. Red checkmarks are placed next to the lines where the product model and product router are registered and used.

```
src > JS app.js > ...
1  const express = require('express');
2  const app = express();
3  app.use(express.json());
4  app.use(express.urlencoded({extended: true}))
5
6  //Configurar conexão
7  const express = require('express')
8  mongoose.connect('mongodb://localhost:27017/mydatabase')
9
10 // Habilita o CORS
11 app.use(function (req, res, next) {
12   res.header('Access-Control-Allow-Origin', '*');
13   res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept, X-access-token');
14   res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
15   next();
16 });
17
18 //registrar os models
19 //criar rotas
20 const index = require('./routes/index')
21 app.use('/', index)
22
23
24 //Registrar as models
25 require('./models/product') ✓
26
27 //Rotas
28 const productRouter = require('./routes/product-route') ✓
29
30 app.use('/products', productRouter) ✓
31
32 module.exports = app;
```

Agora é só ex