



@tiacademybrasil

Back-end com Node.js

CICLO 3 | Profº. Erinaldo

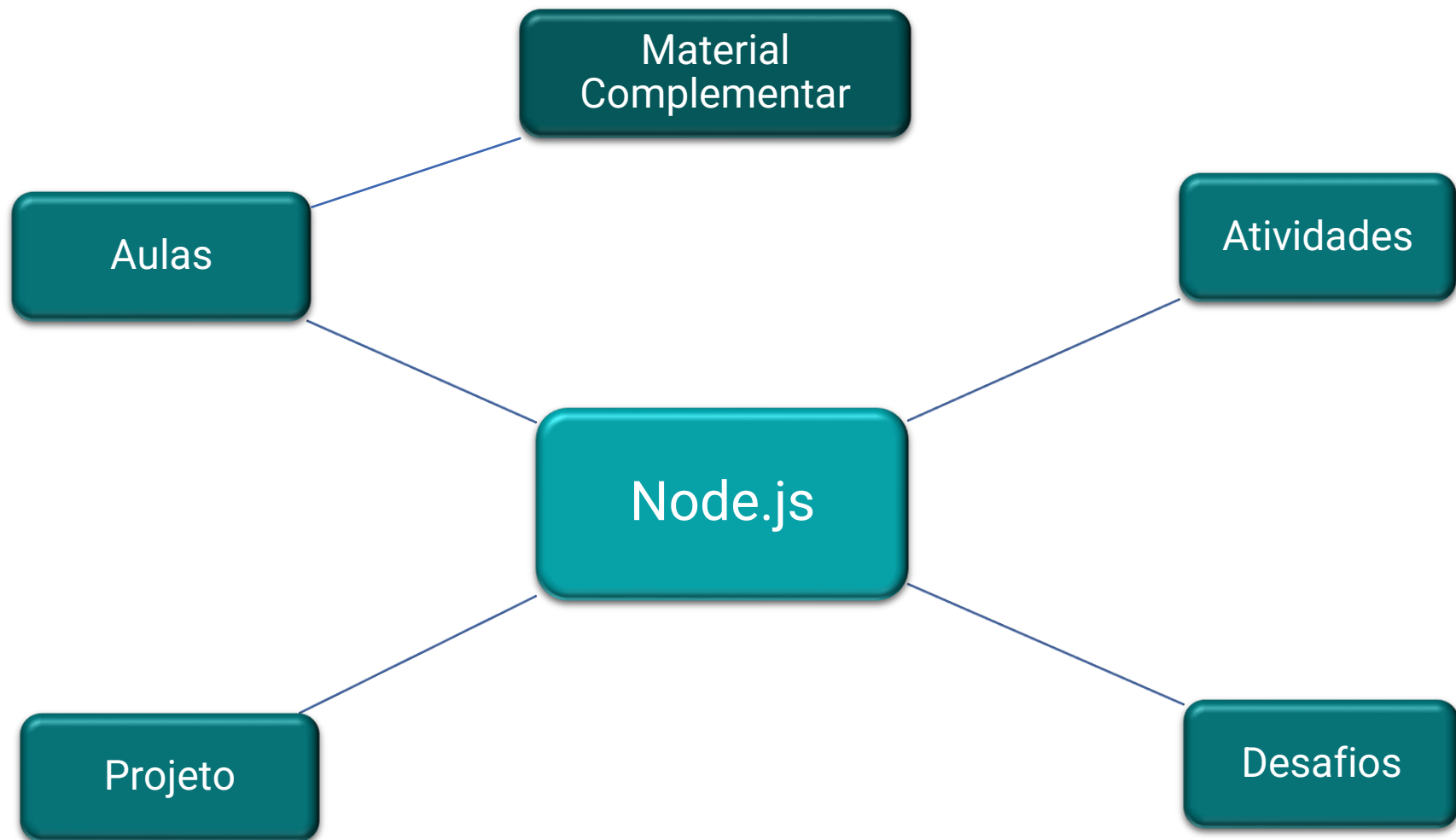
Apresentação



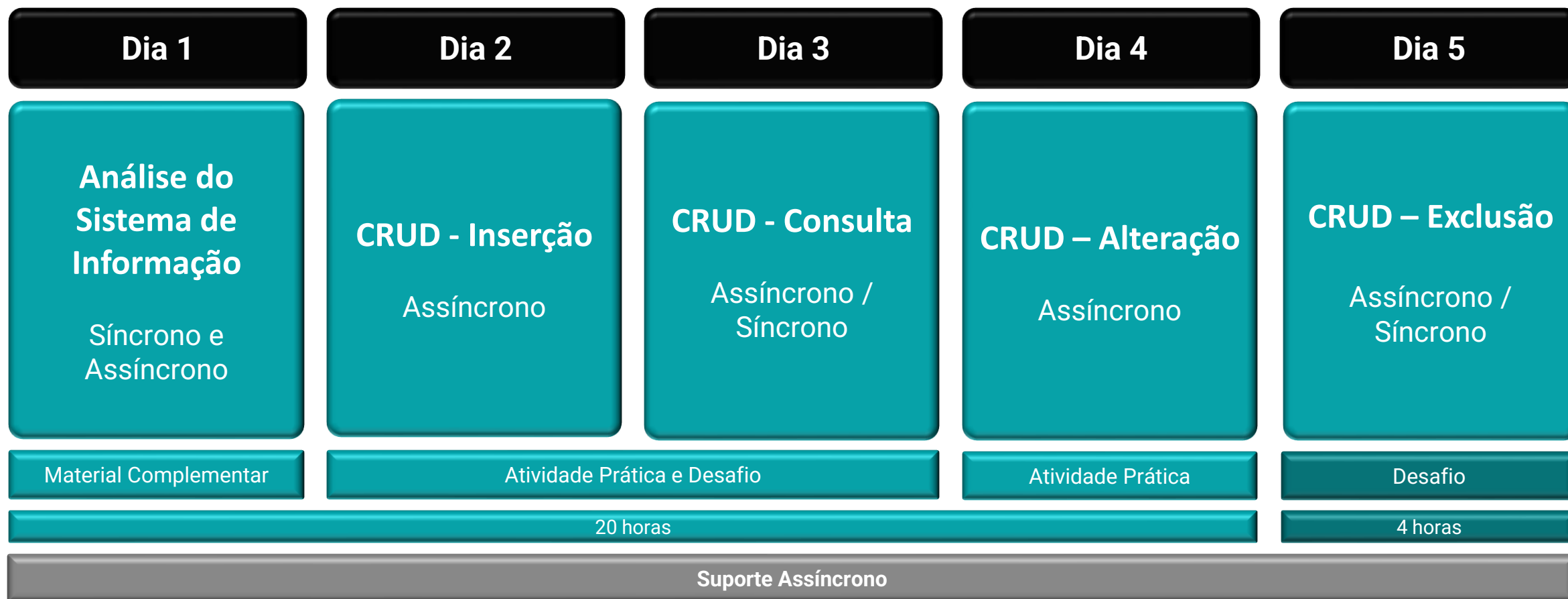
Prof. Me. **Erinaldo Sanches Nascimento**

- Formado em Ciência da Computação (Universidade Santa Cecília - Santos/SP), Especialista em Administração e Desenvolvimento de Banco de Dados (UTFPR - Medianeira/PR), Mestre em Bioinformática (UTFPR - Cornélio Procópio/PR).
- Professor na educação superior nas modalidades presencial e EAD pela Unicesumar (Maringá/PR). Coordena o Ensino Médio Profissional em Informática pela SEEDPR em Sarandi/PR.
- Analista e desenvolvedor web na TICLab.

Estrutura do Curso



Estrutura do Curso



Dia 1: Análise do Sistema de Informação

1. Apresentação do problema
2. Mockup do projeto
3. Estrutura estática do sistema
4. Instalar dependências

TOTAL: 22 horas

3ª semana

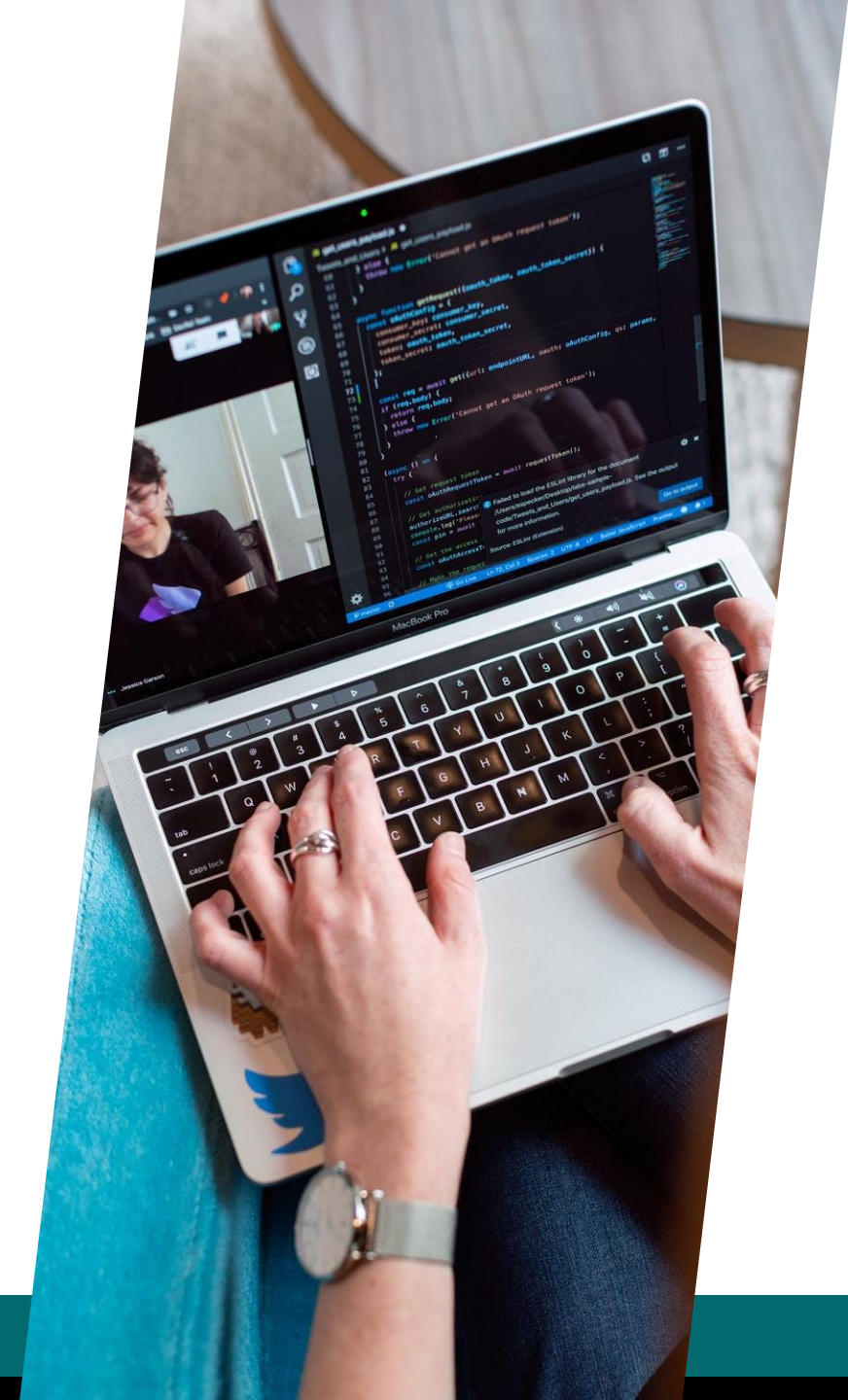


Antes de começar...

Você já aprendeu a criar páginas estáticas e estilizadas com HTML e CSS.

Na sequência você utilizou a linguagem de programação Javascript para desenvolver as estruturas lógicas fundamentais.

Agora chegou o momento de você desenvolver um sistema para Web com a linguagem de programação Nodejs.



Apresentação do problema

Uma empresa do setor de TIC (Tecnologia da Informação e Comunicação) oferece vários serviços para diversos clientes.

Essa empresa deseja ter um sistema que funcione *on-line*, com acesso a internet e que possibilite o controle dos seus clientes, serviços e pedidos.

O controle implica em inserir, remover, alterar e consultar todos os envolvidos na atividade da empresa.



Mockup do projeto

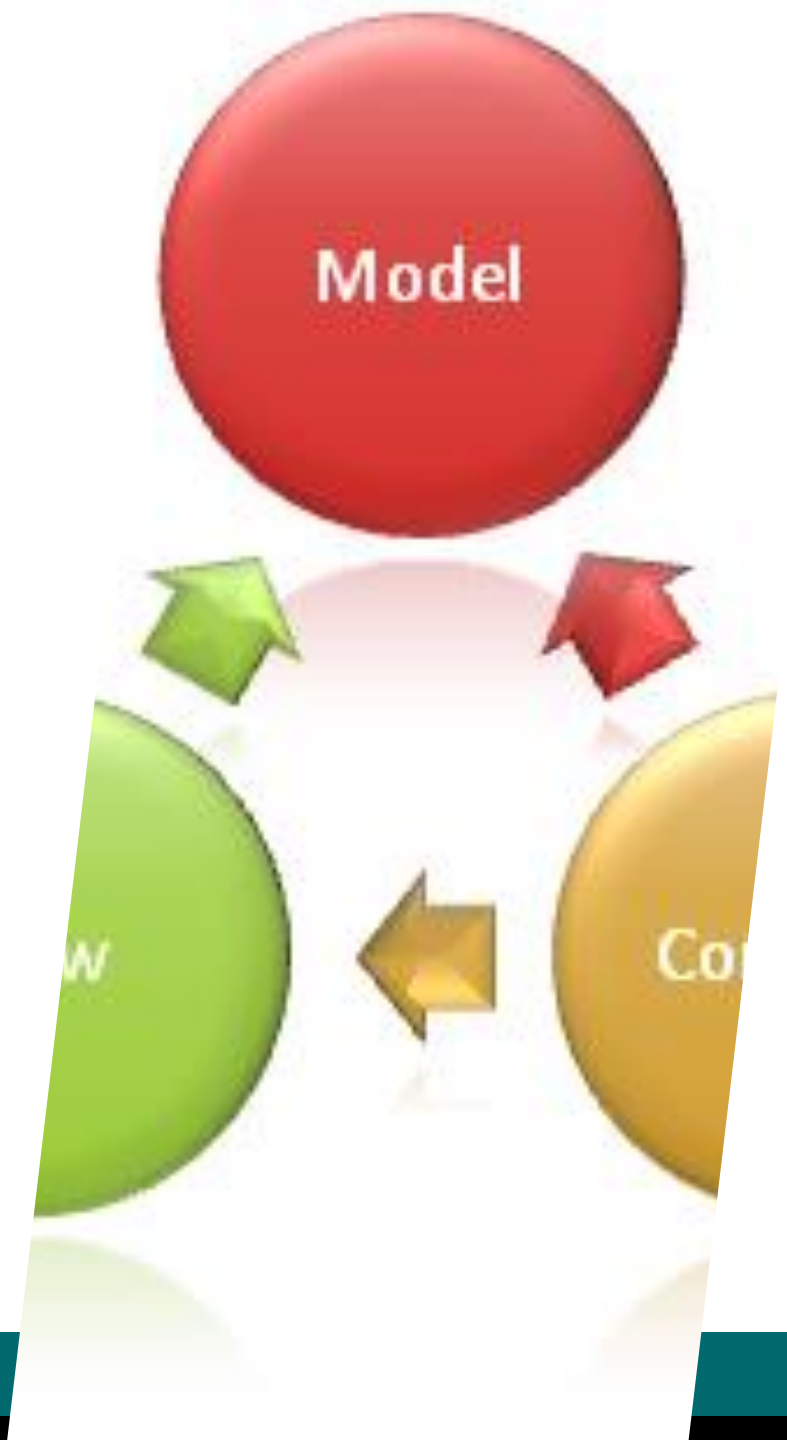
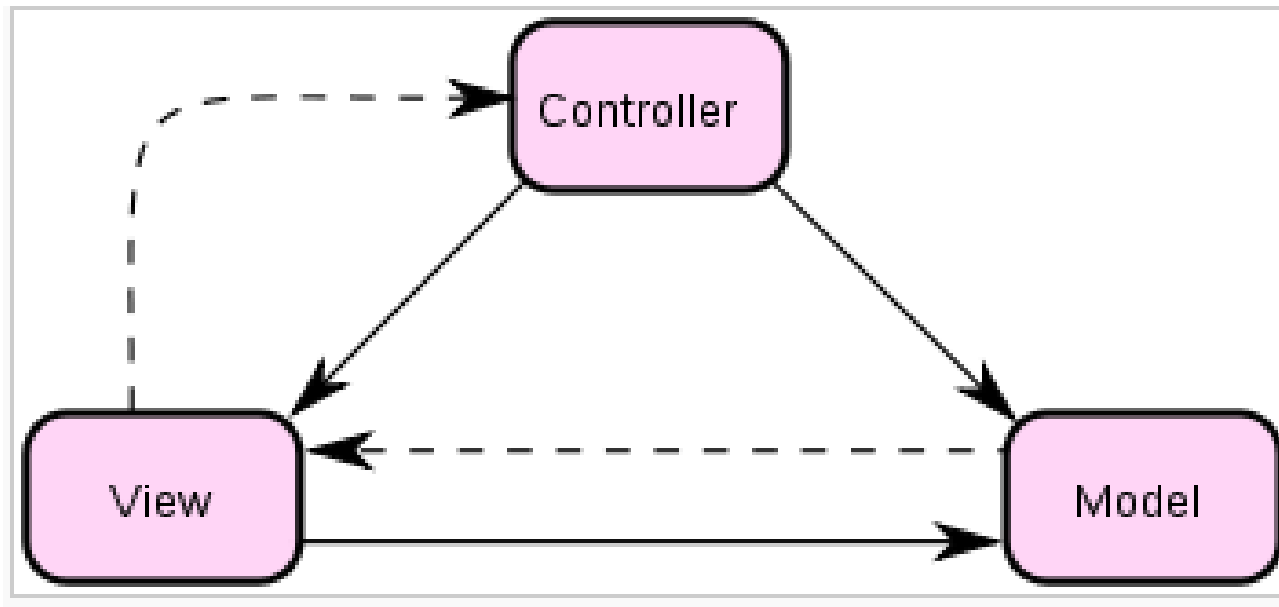
A equipe de designer desenvolveu uma representação da solução do projeto para a empresa.



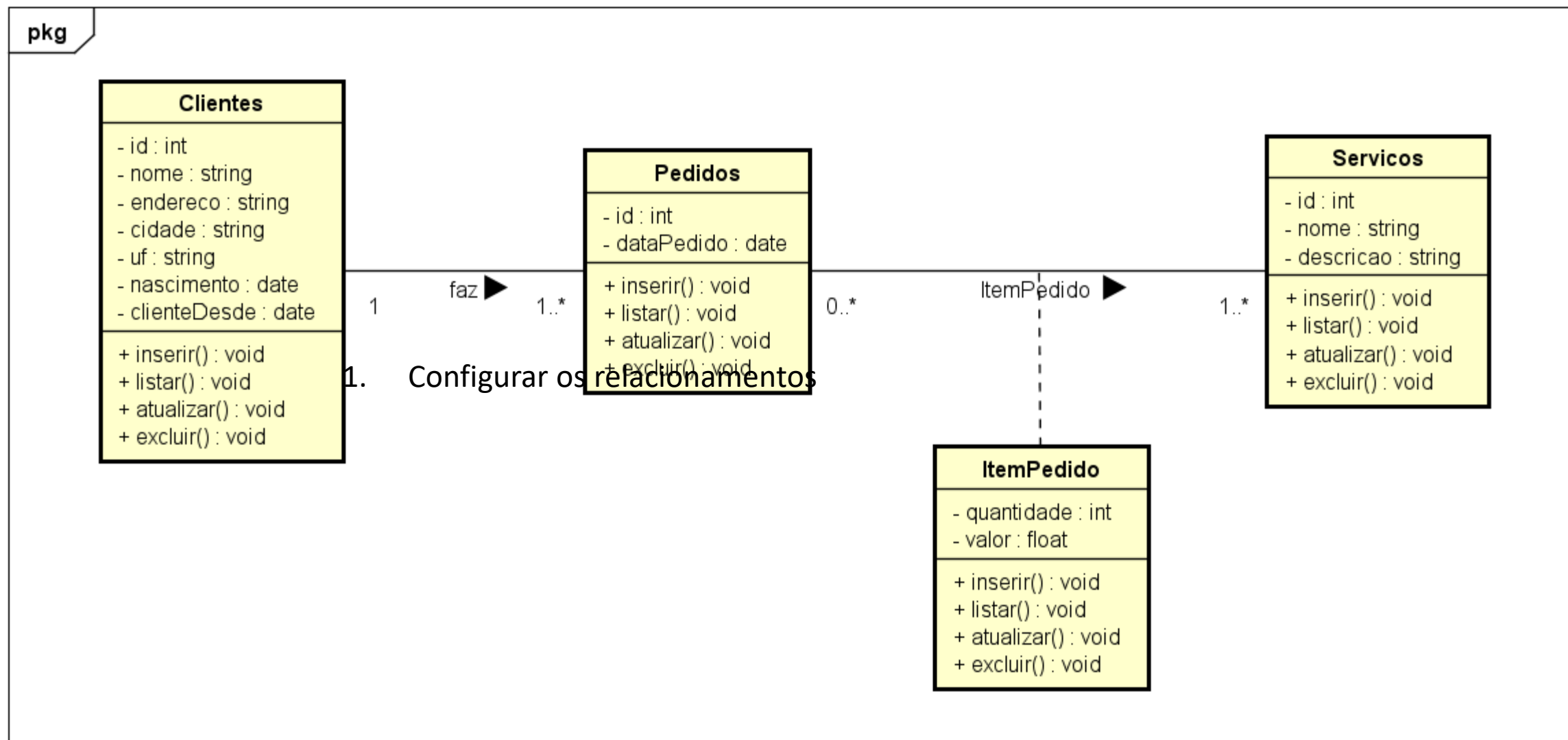
<https://www.figma.com/file/NCol7WCXA63x4R7v8LmNIH/TIAcademyServices?node-id=0%3A1>

Modelagem do projeto

- A equipe de engenharia de software modelou o sistema para funcionar em camadas.
- Uma camada de dados, chamada *Model*.
- Uma camada de exibição, chamada *View*.
- Uma camada de serviços, chamada *Controller*.



Modelagem do projeto



O que estudamos até aqui...

Apresentação do problema

Mockup do projeto

Modelagem do projeto



O que vem depois

Criar a camada de modelo (MODEL)

Configurar os relacionamentos

Migrar para a base de dados



#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

CICLO 3 | Profº. Erinaldo



@tiacademybrasil

Camada de Modelo

Vamos praticar?

Dia 1: Análise do Sistema de Informação

1. Apresentação do problema
2. Mockup do projeto
3. Modelagem do projeto

TOTAL: 22 horas

3ª semana



Dia 1: Análise do Sistema de Informação

1. Instalar as dependências
2. Criar a base de dados
3. Criar a camada de modelo (MODEL)

TOTAL: 22 horas

3ª semana



Criar a camada de modelo (MODEL)

1. Criar o diretório da aplicação (`ciclo3`).
2. Acessar o diretório e executar `npm init`.
3. Instalar o framework **Express** no diretório da aplicação.
 - a) Acesse o site <https://expressjs.com/pt-br/>.
 - b) Copie o comando `npm install express -save`.
4. Instalar o módulo **Cors**.
 - a) Acesse o site <http://expressjs.com/en/resources/middleware/cors.html>.
 - b) Execute o comando de instalação `npm install cors`.



Criar a camada de modelo (MODEL)

5. Instalar o framework **Sequelize**.

- Acessar o site <https://sequelize.org/>.
- Copiar e executar o comando de npm `install --save sequelize`.
- Acessar a documentação mais recente no endereço <https://sequelize.org/master/manual/getting-started.html>.
- Copie e execute o comando `npm install --save mysql2` para instalar o módulo MySQL.

6. Instalar o módulo **Sequelize Cli**.

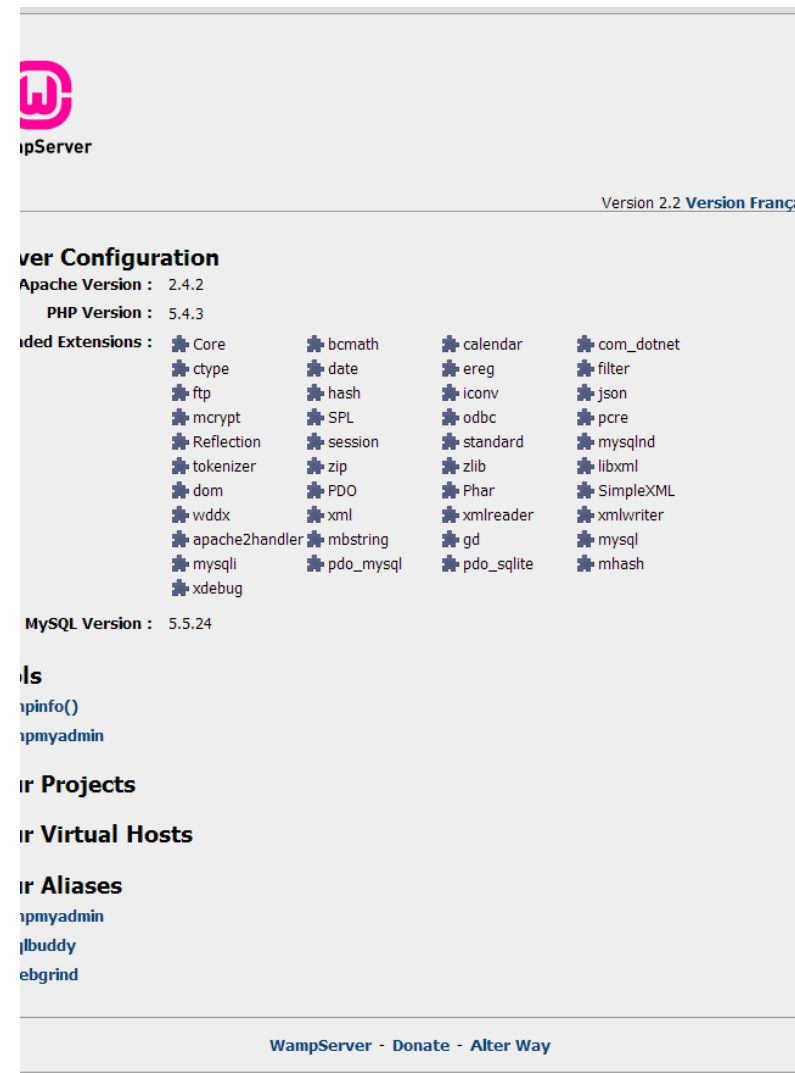
- Acessar o site <https://sequelize.org/master/manual/migrations.html>.
- Copiar e executar o comando `npm install --save-dev sequelize-cli`.
- Crie a estrutura de pastas com o comando `npx sequelize-cli init`.



Criar a camada de modelo (MODEL)



7. Conectar o WampServer.
8. Abrir o *localhost* (<http://localhost/>).
9. Abrir o phpMyAdmin
(<http://localhost/phpmyadmin/>).
 - a) O usuário padrão é `root`.
 - b) Sem senha.
 - c) Crie uma nova base de dados `bdciclo3`.



Criar a camada de modelo (MODEL)

phpMyAdmin

Servidor atual: MySQL

Base de Dados SQL Estado Contas de utilizador Exportar Importar

Base de Dados

Criar base de dados

turma1 utf8_unicode_ci Criar

Base de Dados	Agrupamento (Collation)	Acções
<input type="checkbox"/> information_schema	utf8_general_ci	Verificar Privilégios
<input type="checkbox"/> mysql	latin1_swedish_ci	Verificar Privilégios
<input type="checkbox"/> performance_schema	utf8_general_ci	Verificar Privilégios
<input type="checkbox"/> sys	utf8_general_ci	Verificar Privilégios

Total: 4

☐ Marcar todos Com os seleccionados: [Elimina](#)

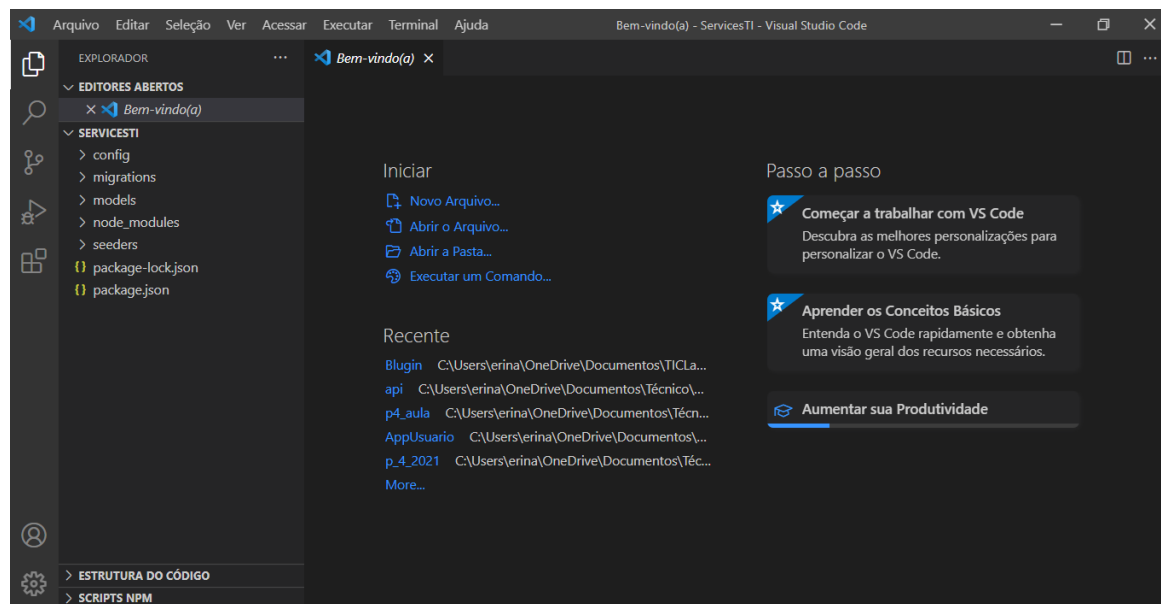
⚠ Nota: Activar as estatísticas aqui pode causar um grande volume de tráfego entre os servidores web e MySQL.

- [Activar estatísticas](#)

Criar a camada de modelo (MODEL)

Vamos criar o
nosso modelo
de dados?

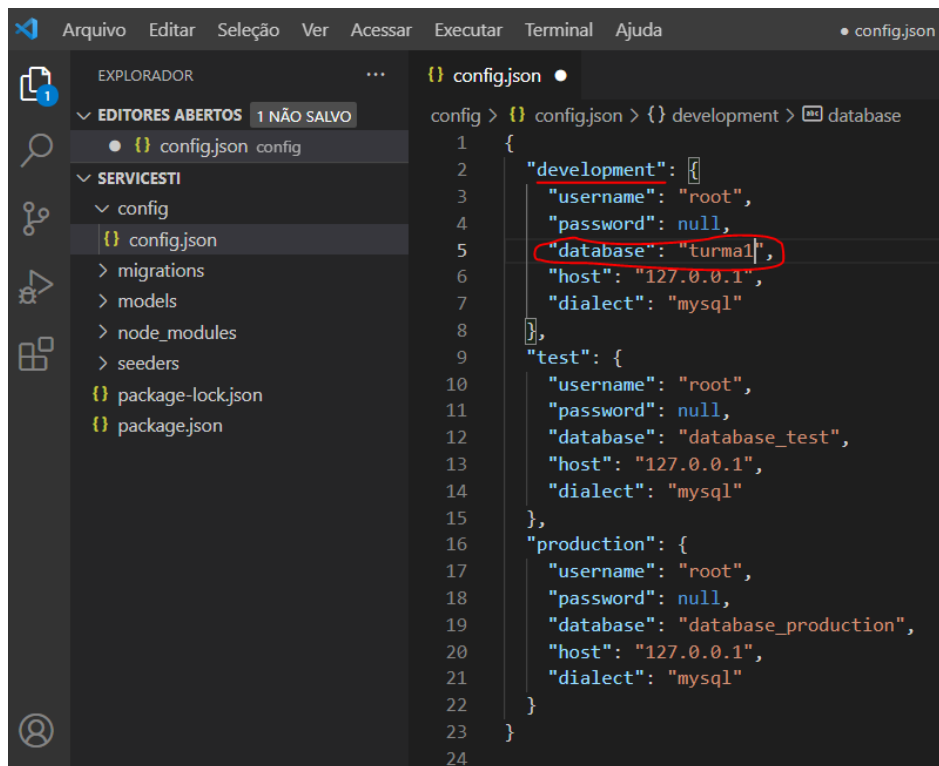
10. Abra o Visual Code Studio.



Criar a camada de modelo (MODEL)

11. Abra a pasta config e o arquivo config.json.

a) Na lista de dados development, altere o valor da chave database para bdciclo3.



```
config > {} config.json > {} development > database
1 {
2   "development": {
3     "username": "root",
4     "password": null,
5     "database": "turmal",
6     "host": "127.0.0.1",
7     "dialect": "mysql"
8   },
9   "test": {
10    "username": "root",
11    "password": null,
12    "database": "database_test",
13    "host": "127.0.0.1",
14    "dialect": "mysql"
15  },
16  "production": {
17    "username": "root",
18    "password": null,
19    "database": "database_production",
20    "host": "127.0.0.1",
21    "dialect": "mysql"
22  }
23 }
```



Criar a camada de modelo (MODEL)

12. Abra um novo terminal no VSCode.

13. Crie o modelo clientes.

- a) Abra o site
<https://sequelize.org/master/manual/migrations.html>.
- b) Copie (mas ainda não execute) o comando `npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string`.
- c) Altere o nome do modelo para `Cliente` e os atributos e tipos de dados conforme a indicação a seguir:



Criar a camada de modelo (MODEL)

Cientes
- id : int - nome : string - endereco : string - cidade : string - uf : string - nascimento : date - clienteDesde : date
+ inserir() : void + listar() : void + atualizar() : void + excluir() : void

- Verifique se a pasta `models` contém, entre outros, o arquivo `cliente.js`.
- A pasta `migrations` deve conter um arquivo que começa com uma numeração e termina com `create-cliente.js`.



Criar a camada de modelo (MODEL)

Agora vai uma atividade...

Refaça a etapa 13 e crie o modelo para Serviços.

Servicos
- id : int - nome : string - descricao : string
+ inserir() : void + listar() : void + atualizar() : void + excluir() : void

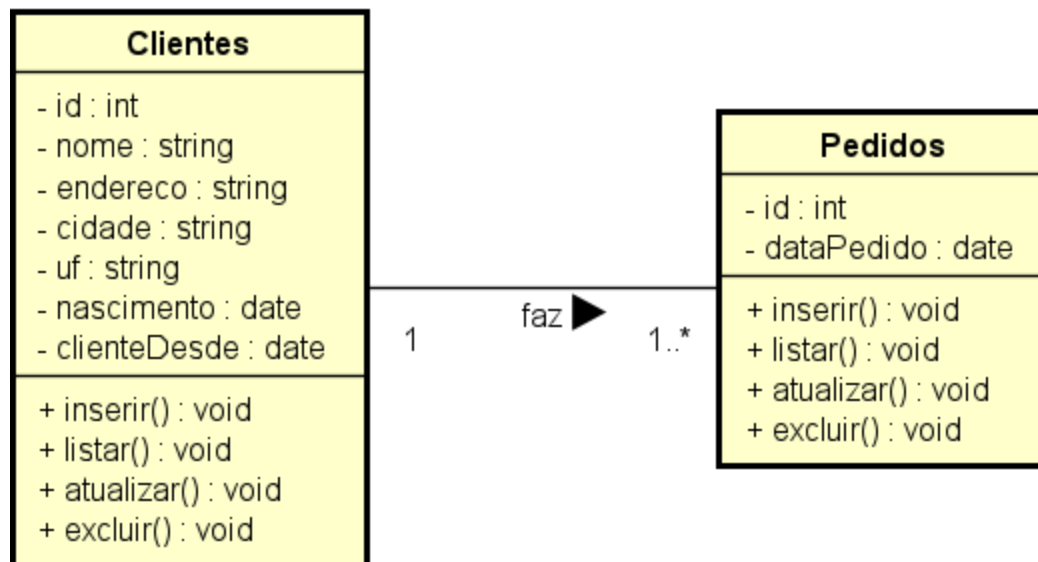
Ah... o campo `id` é criado automaticamente. Então os atributos são apenas `nome` e `descricao`.



Criar a camada de modelo (MODEL)

Agora vamos fazer juntos...

Pedidos vai utilizar dados de Clientes.



No endereço do manual do Sequelize a seguir, <https://sequelize.org/master/manual/model-basics.html#data-types>, traz os tipos de dados.



Criar a camada de modelo (MODEL)

Muito bem...

Agora vamos configurar os relacionamentos e migrar para a nossa base de dados.

Antes... dê uma olhadinha em

<https://sequelize.org/master/manual/assocs.html>.



O que estudamos até aqui...

Instalar as dependências

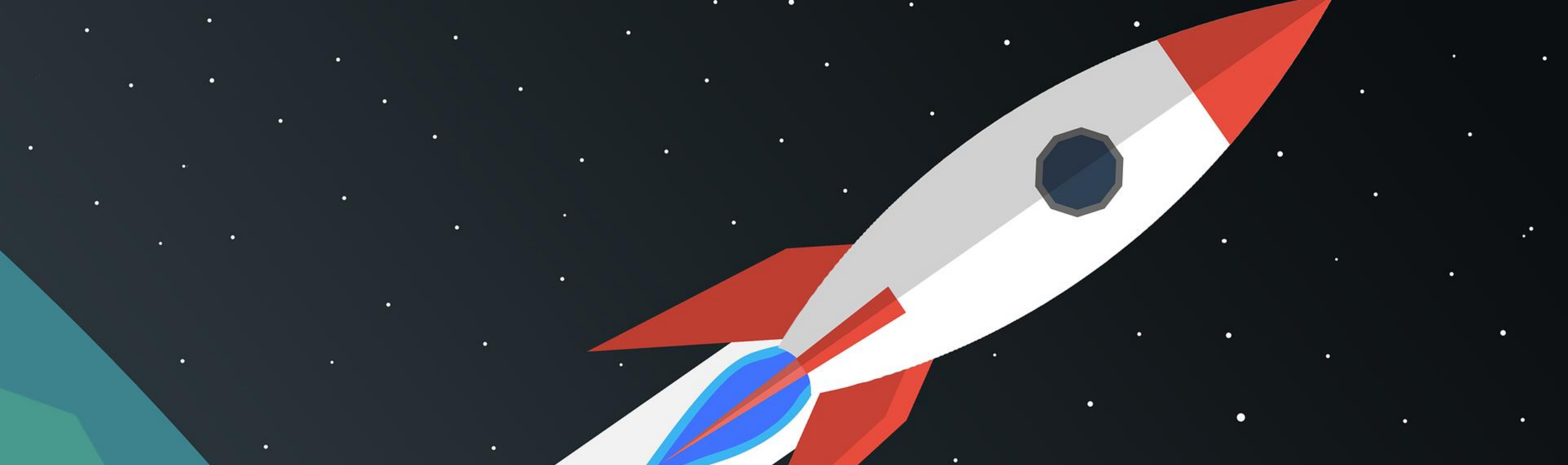
Criar a base de dados

Criar a camada de modelo (MODEL)

O que vem depois

Configurar os relacionamentos
Migrar para a base de dados





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 3 | Profº. Erinaldo



@tiacademybrasil

Camada de Modelo

Vamos praticar?

Dia 1: Análise do Sistema de Informação

1. Instalar as dependências
2. Criar a base de dados
3. Criar a camada de modelo (MODEL)

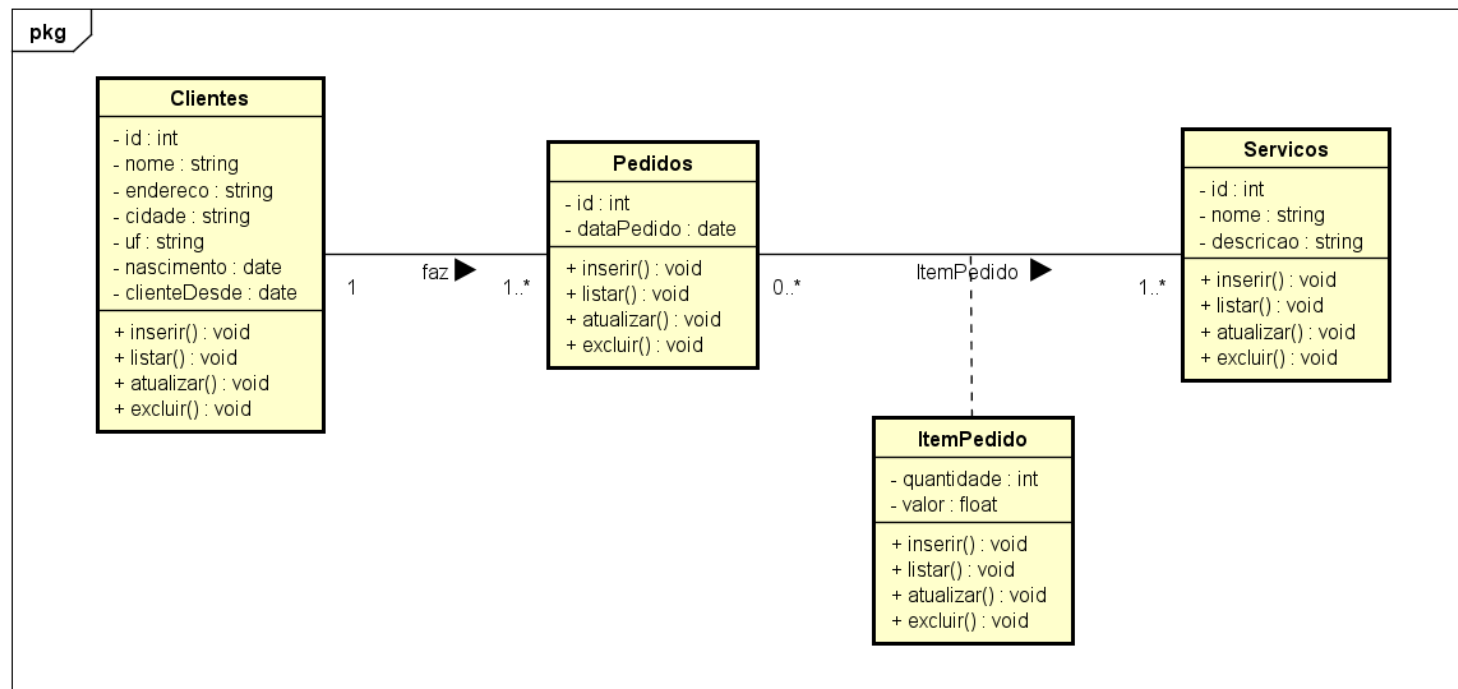
TOTAL: 22 horas

3ª semana



Dia 1: Análise do Sistema de Informação

1. Configurar os relacionamentos



TOTAL: 22 horas

3ª semana

Configurar os relacionamentos

Vamos iniciar pelo cliente.

- Abra o arquivo `cliente.js` na pasta `models`.
- Altere a associação conforme a seguir:

```
models > JS cliente.js > <unknown> > exports > Cliente > associate
1  'use strict';
2  const {
3    Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    class Cliente extends Model {
7      /**
8       * Helper method for defining associations.
9       * This method is not a part of Sequelize lifecycle.
10      * The `models/index` file will call this method automatically.
11      */
12     static associate(models) {
13       Cliente.hasMany(Pedido);
14     }
15   };
16 }
```



Configurar os relacionamentos

Na sequência são os serviços.

Consegue fazer sozinho(a)?

Vamos lá...

Um serviço pode estar em muitos pedidos.

Agora vamos falar dos pedidos...

- Um pedido pertence a um cliente.
- Um pedido possui um único serviço.



Configurar os relacionamentos

```
models > JS pedido.js > <unknown> > exports
1  'use strict';
2  const {
3    Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    class Pedido extends Model {
7      /**
8       * Helper method for defining associations.
9       * This method is not a part of Sequelize lifecycle.
10      * The `models/index` file will call this method automatically.
11      */
12     static associate(models) {
13       Pedido.belongsTo(Cliente);
14       Pedido.belongsTo(Servico);
15     }
16   };
17 }
```

Falta pouco para gerarmos nossa base de dados...



O que estudamos até aqui...

Configurar os relacionamentos

- Muitos para um
- Muitos para muitos

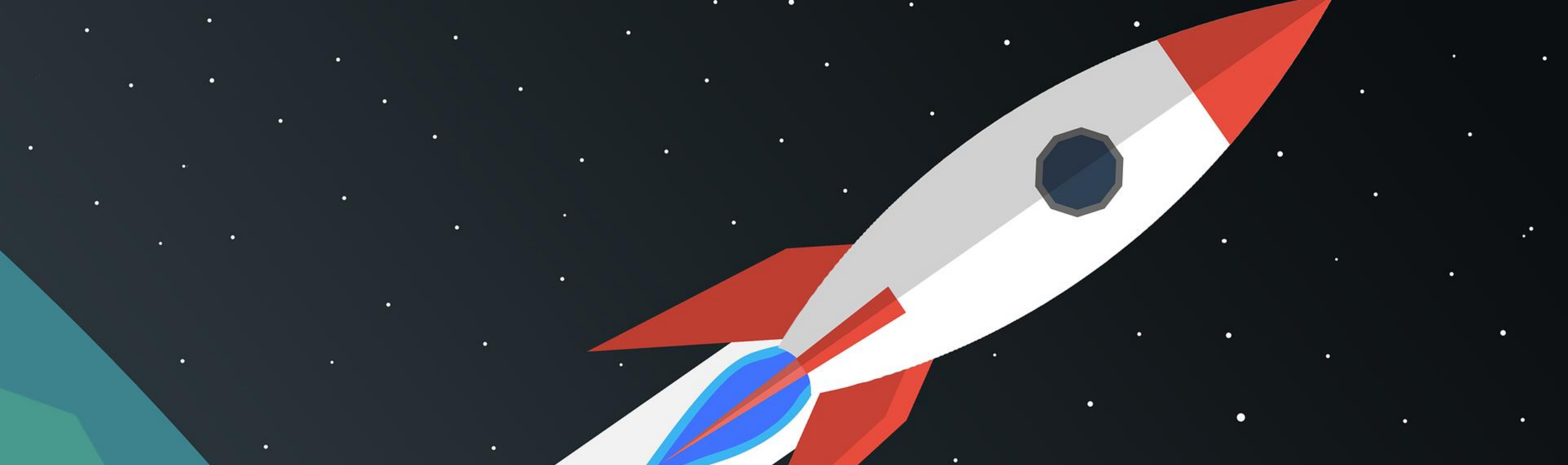


O que vem depois

Migrar para a base de dados

- Tipos de dados
- Chave primária
- Chave estrangeira





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 4 | Profº. Erinaldo



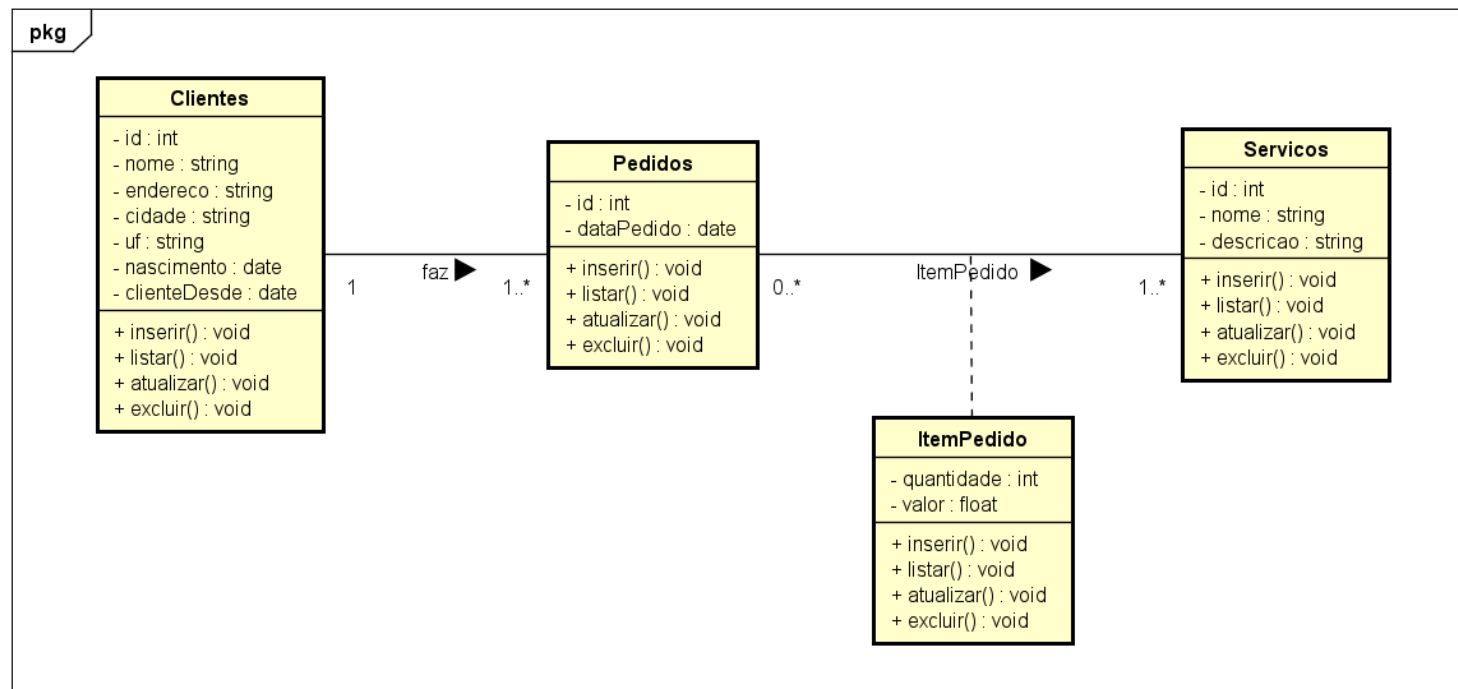
@tiacademybrasil

Camada de Modelo

Vamos praticar?

Dia 1: Análise do Sistema de Informação

1. Configurar os relacionamentos



TOTAL: 22 horas

3ª semana

Dia 1: Análise do Sistema de Informação

1. Migrar para a base de dados

- Tipos de dados
- Chave primária
- Chave estrangeira

TOTAL: 22 horas

3ª semana



Migrar para a base de dados

Na pasta `migrations` precisamos definir o conceito de chave estrangeira para refletir a associação entre cliente e pedido e pedido e serviço.

```
18     clienteId: {
19       type: Sequelize.INTEGER,
20       references: {
21         model: 'clientes',
22         key: 'id'
23       },
24       onDelete: 'CASCADE',
25       onUpdate: 'CASCADE'
26     },
27     servicoId: {
28       type: Sequelize.INTEGER,
29       references: {
30         model: 'servicos',
31         key: 'id'
32       },
33       onDelete: 'CASCADE',
34       onUpdate: 'CASCADE'
35     },
```



Migrar para a base de dados

Está acabando...

Salve tudo, atualizando os arquivos alterados.

Acesse o endereço

<https://sequelize.org/master/manual/migrations.html#running-migrations>.

No terminal, no diretório do projeto copie e execute comando `npx sequelize-cli db:migrate`.

Verifique a sua base de dados no <http://localhost/phpmyadmin>.



O que estudamos até aqui...

Criar a camada de modelo (MODEL)

Configurar os relacionamentos

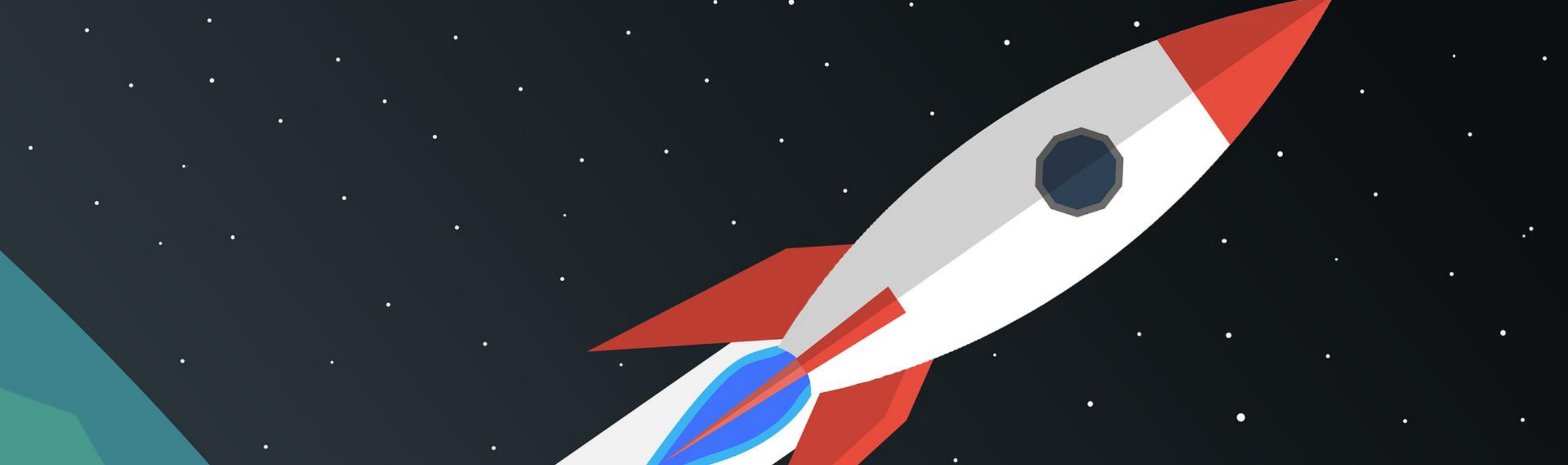
Migrar para a base de dados



O que vem depois

Agora que temos a base de dados pronta, vamos implementar o CRUD da nossa aplicação.





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 5 | Profº. Erinaldo



@tiacademybrasil

CRUD - Inserção

Vamos praticar?

Dia 1: Análise do Sistema de Informação

Migrar para a base de dados

- Tipos de dados
- Chave primária
- Chave estrangeira

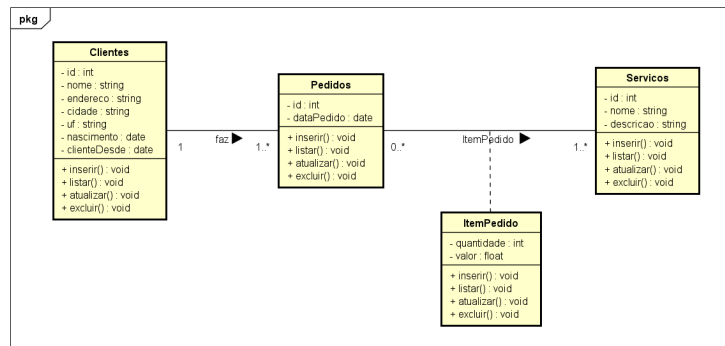


Tabela	Acções	Registos	Tipo	Agrupamento (Collation)	Tamanho	Suspensão
<input type="checkbox"/> clientes	★ Procurar Estrutura Pesquisar Inserir Limpar Eliminar	0	InnoDB	utf8_unicode_ci	16.0 KB	-
<input type="checkbox"/> itempedidos	★ Procurar Estrutura Pesquisar Inserir Limpar Eliminar	0	InnoDB	utf8_unicode_ci	32.0 KB	-
<input type="checkbox"/> pedidos	★ Procurar Estrutura Pesquisar Inserir Limpar Eliminar	0	InnoDB	utf8_unicode_ci	32.0 KB	-
<input type="checkbox"/> sequelizemeta	★ Procurar Estrutura Pesquisar Inserir Limpar Eliminar	4	InnoDB	utf8_unicode_ci	32.0 KB	-
<input type="checkbox"/> servicos	★ Procurar Estrutura Pesquisar Inserir Limpar Eliminar	0	InnoDB	utf8_unicode_ci	16.0 KB	-
5 tabelas	Soma	4	MyISAM	utf8_unicode_ci	128.0 KB	0 Bytes

TOTAL: 22 horas

3ª semana

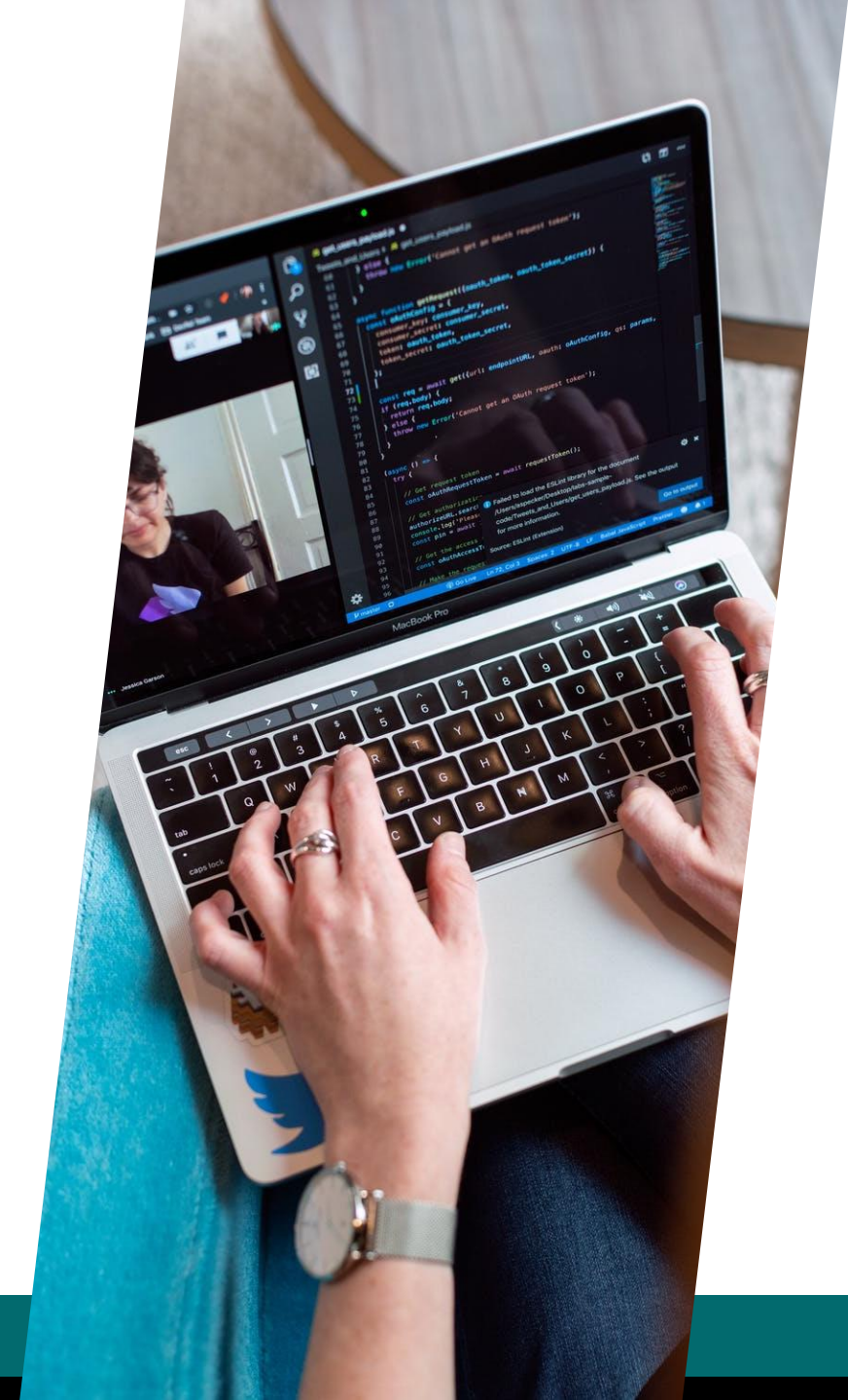
Antes de começar...

Você criou um novo projeto utilizando o Nodejs.

Instalou as principais dependências para criar o modelo de dados do sistema.

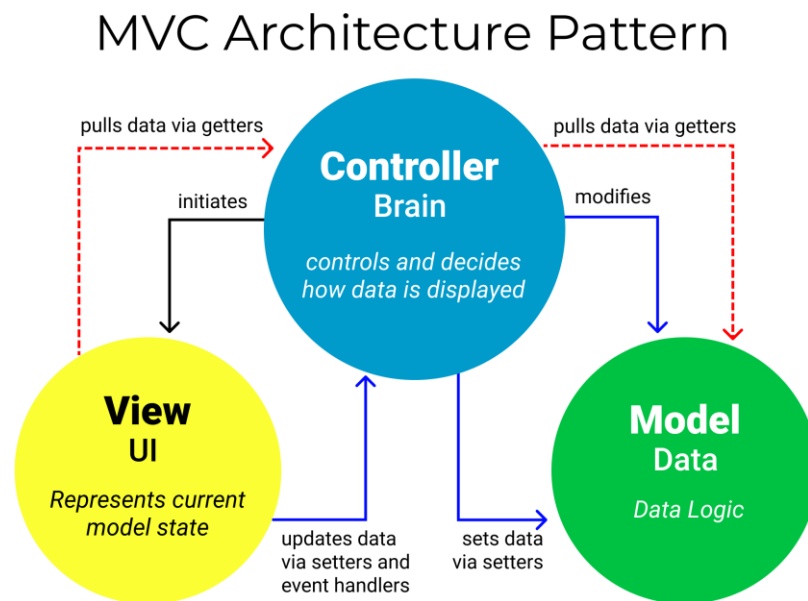
Criou o modelo de dados e exportou para o servidor de dados da aplicação.

Agora chegou o momento de começar a criar o CRUD (Create, Retrieve, Update e Delete). Hoje é dia de implementar as inserções.



Dia 2: CRUD - Inserção

Criação da camada de controle da aplicação.



TOTAL: 22 horas

3ª semana

Apresentação do problema

No padrão MVC (Model – View – Controller) de desenvolvimento, você já implementou o *Model*.

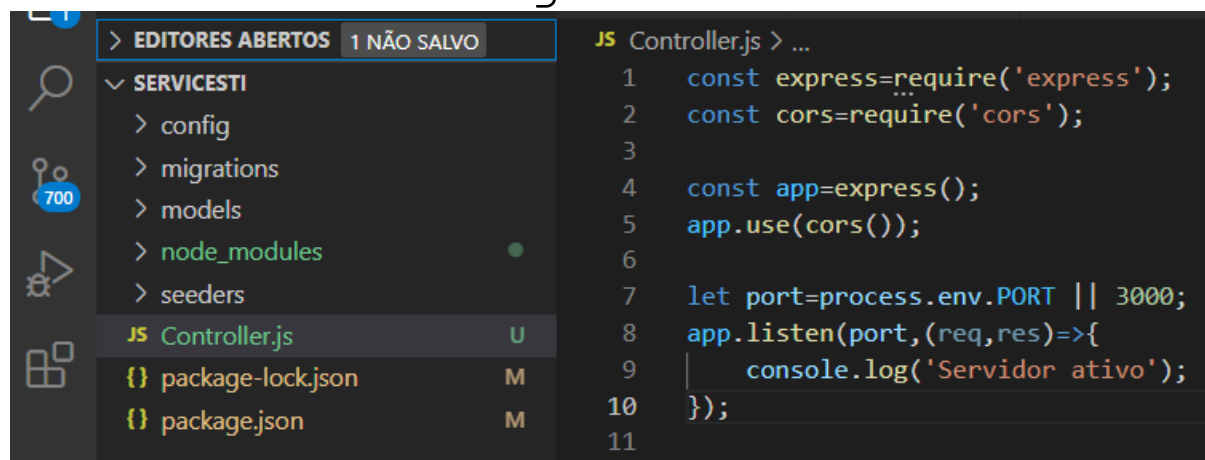
Nessa aula você vai começar a criar o *Controller*, a parte responsável pelos serviços da aplicação.

Create promove a operação de inserção. De modo prático, o *create* cria ou adiciona novas entradas na aplicação.



Criação da camada de controle da aplicação

1. Abra a aplicação no VSCode.
2. Acesse o endereço <https://www.npmjs.com/package/nodemon> e execute `npm install --save-dev nodemon` para a instalação do módulo nodemon.
3. Na raiz da aplicação crie o arquivo `Controller.js`.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project named 'SERVICESTI' with folders like 'config', 'migrations', 'models', 'node_modules', and 'seeders'. The 'Controller.js' file is selected. The main editor area shows the content of 'Controller.js' with the following code:

```
JS Controller.js > ...
1  const express=require('express');
2  const cors=require('cors');
3
4  const app=express();
5  app.use(cors());
6
7  let port=process.env.PORT || 3000;
8  app.listen(port,(req,res)=>{
9    console.log('Servidor ativo');
10 });
11
```



Criação da camada de controle da aplicação

4. Execute no terminal o comando `nodemon Controller.js`.

```
TERMINAL  PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO

[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node Controller.js`
[nodemon] restarting due to changes...
[nodemon] starting `node Controller.js`
Servidor ativo
```



Criação da camada de controle da aplicação

5. Criar uma rota `get` simples do `Express` para responder na página principal da aplicação.

```
JS Controller.js > ...
1  const express=require('express');
2  const cors=require('cors');
3
4  const app=express();
5  app.use(cors());
6
7  app.get('/', function (req, res) {
8    res.send('Hello World!');
9  });
10
11 let port=process.env.PORT || 3000;
12 app.listen(port,(req,res)=>{
13   console.log('Servidor ativo');
14 });
15
```



Criação da camada de controle da aplicação

6. Abra a aplicação no navegador digitando o endereço `localhost:3001`.

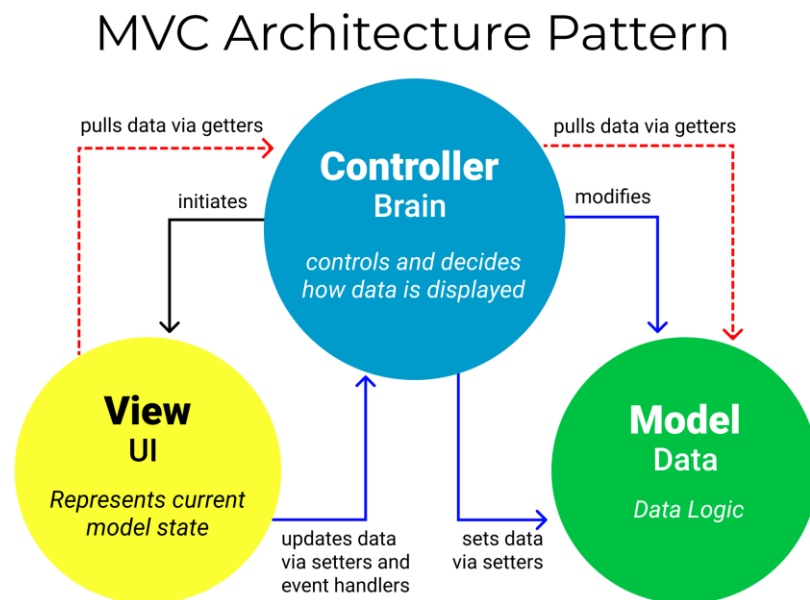
EXERCÍCIO

1. Crie uma rota para clientes, serviços e pedidos. Em cada rota crie e apresente um texto diferente.
2. Para executar cada rota digite `localhost:3001/rota`.



O que estudamos até aqui...

Criação da camada de controle da aplicação.



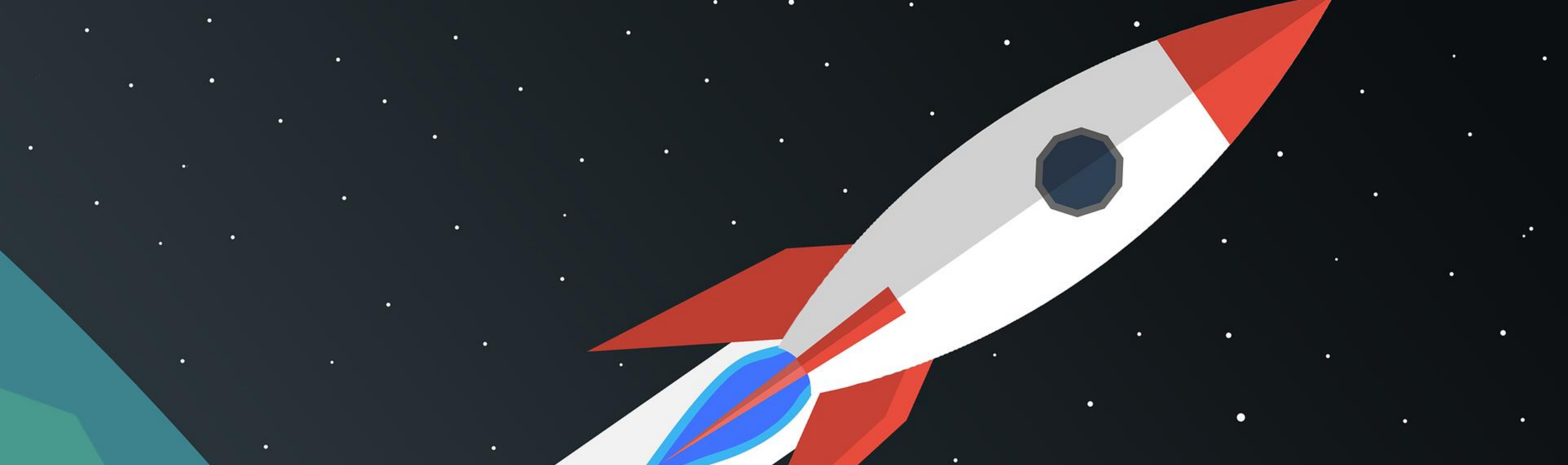
O que vem depois

Criação do método *create*

Simular requisições externas

Verificar se foi cadastrado com sucesso





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 6 | Profº. Erinaldo



@tiacademybrasil

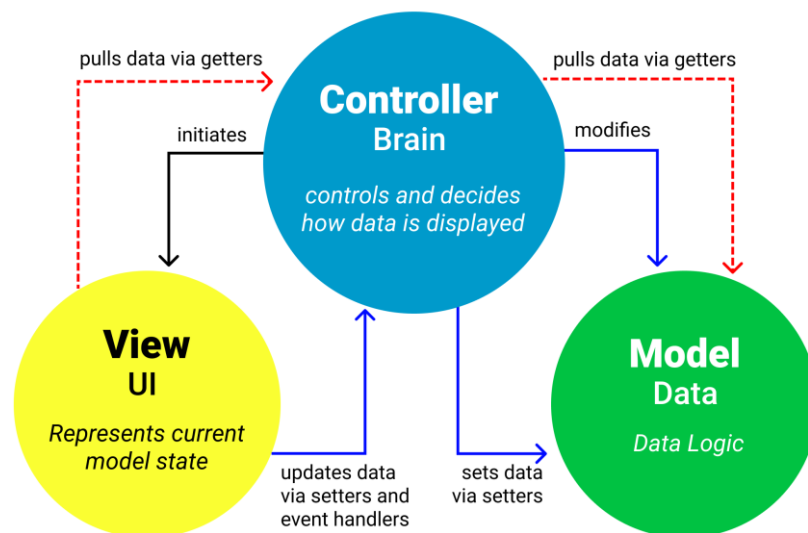
CRUD - Inserção

Vamos praticar?

Dia 2: CRUD - Inserção

Criação da camada de controle da aplicação.

MVC Architecture Pattern



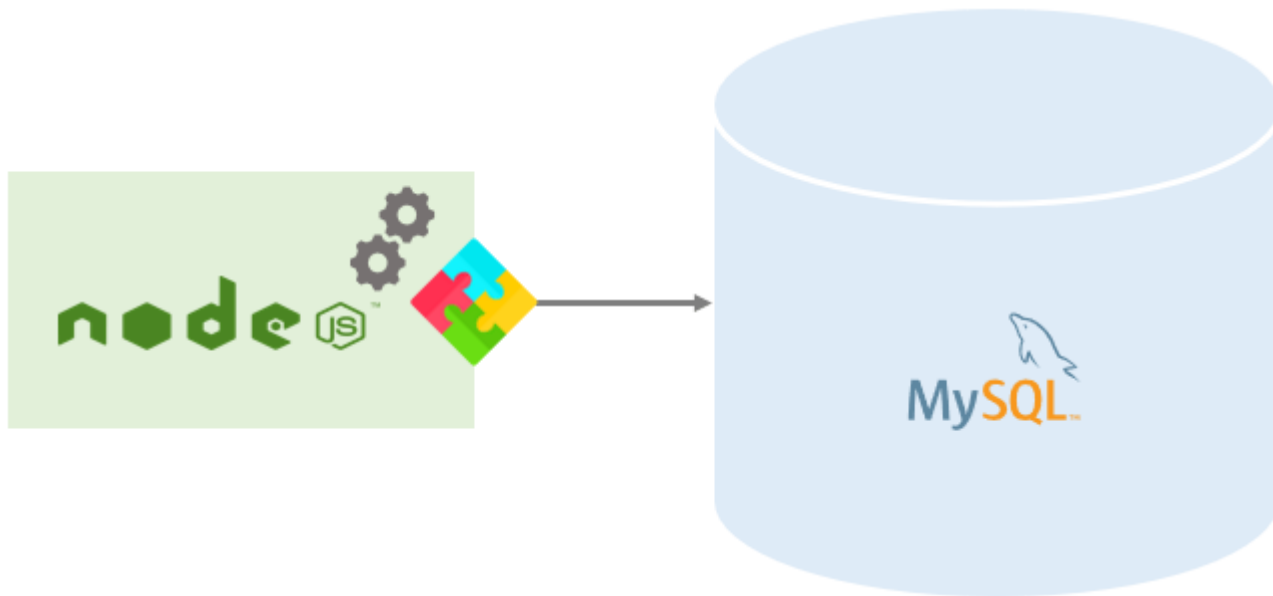
TOTAL: 22 horas

3ª semana



Dia 2: CRUD - Inserção

Criação do método *create*



TOTAL: 22 horas

3ª semana



Criação do método create

1. No Controller.js crie uma constante associada a camada de modelo (models).

```
JS Controller.js > ...
1  const express=require('express');
2  const cors=require('cors');
3
4  const models=require('./models');
5  |
6  const app=express();
7  app.use(cors());
8
9  app.get('/', function (req, res) {
10     res.send('Hello World!');
11 });
12
13 app.get('/clientes', function (req, res) {
14     res.send('Sem bem-vindo a Services TI!');
15 });
```



Criação do método create

2. Crie uma para cada objeto.

```
JS Controller.js > ...
1  const express=require('express');
2  const cors=require('cors');
3
4  const models=require('./models');
5
6  const app=express();
7  app.use(cors());
8
9  let cliente=models.Cliente;
10 let servico=models.Servico;
11 let pedido=models.Pedido;
12
13
14 app.get('/', function (req, res) {
15   res.send('Hello World!');
16 });
```



Criação do método create

3. Altere a rota `get` de acordo com o documento <https://sequelize.org/master/manual/model-querying-basics.html>.

```
13 ✓ app.get('/servicos', async (req, res) => {  
14 ✓   let create=await servico.create({  
15     nome: "HTML/CSS",  
16     descricao: "Páginas estáticas estilizadas",  
17     createAt: new Date(),  
18     updateAt: new Date()  
19   });  
20   res.send('Serviço criado com sucesso!');  
21 });
```

4. Execute a aplicação com `nodemon Controller.js` e acesse a rota `http://localhost:3000/servicos`.



Criação do método create

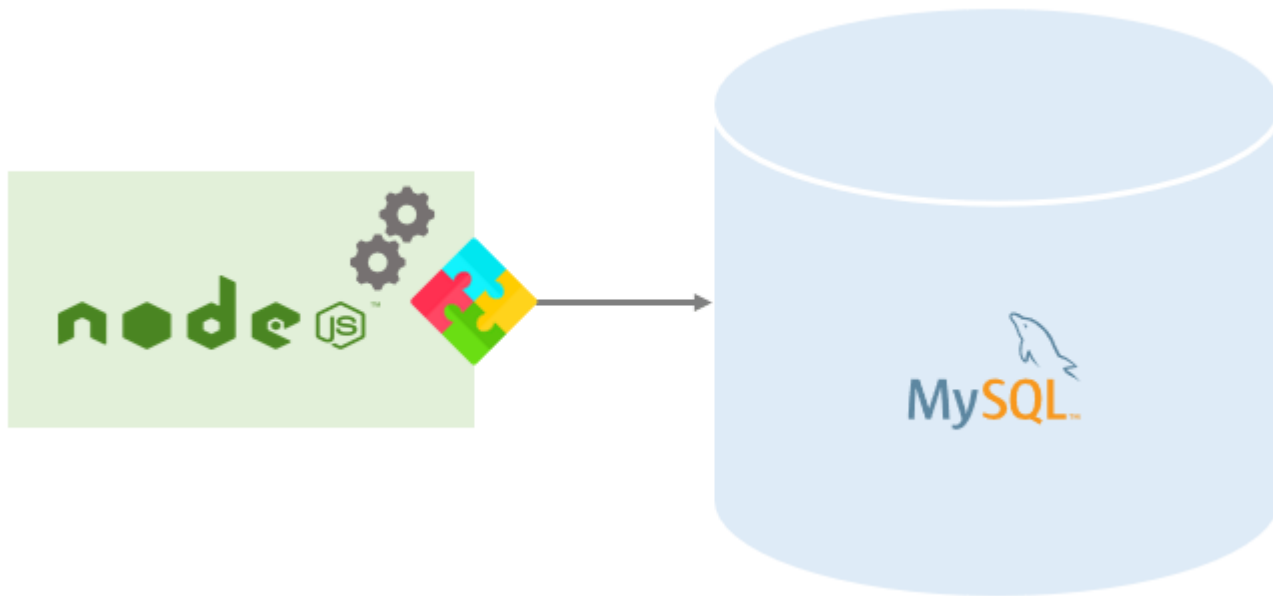
EXERCÍCIO

1. Utilize a mesma rota para criar um novo registro de serviço (Nodejs, Desenvolvimento de aplicação back-end).
2. Crie um registro para um novo cliente.
3. Crie um registro para um novo pedido.
4. Insira um registro como item desse novo pedido.



O que estudamos até aqui...

Criação do método *create*

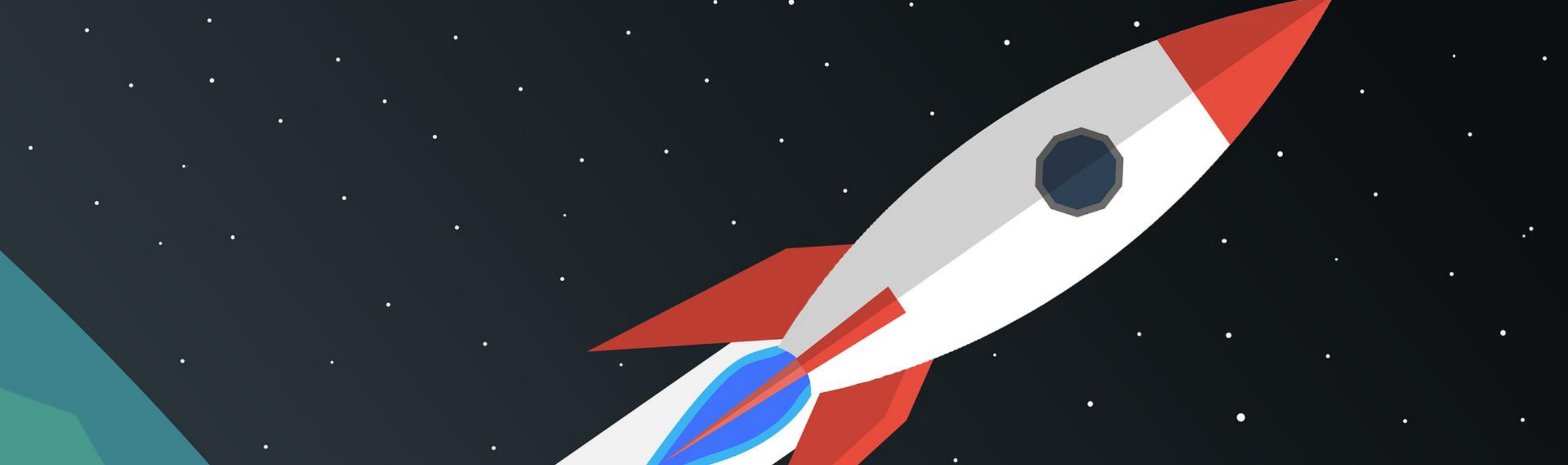


O que vem depois

Simular requisições externas

Verificar se foi cadastrado com sucesso





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 7 | Profº. Erinaldo



@tiacademybrasil

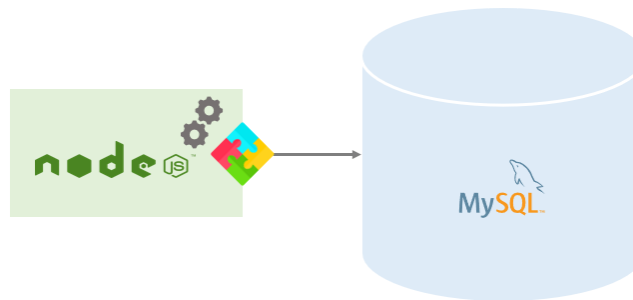
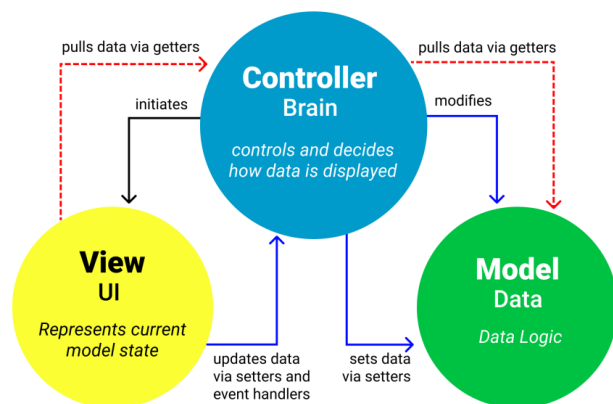
CRUD - Inserção

Vamos praticar?

Dia 2: CRUD - Inserção

1. Criação da camada de controle da aplicação.
2. Criação do método *create*

MVC Architecture Pattern



TOTAL: 22 horas

3ª semana

Dia 2: CRUD - Inserção

Simular requisições externas

Verificar se foi cadastrado com sucesso



POSTMAN

TOTAL: 22 horas

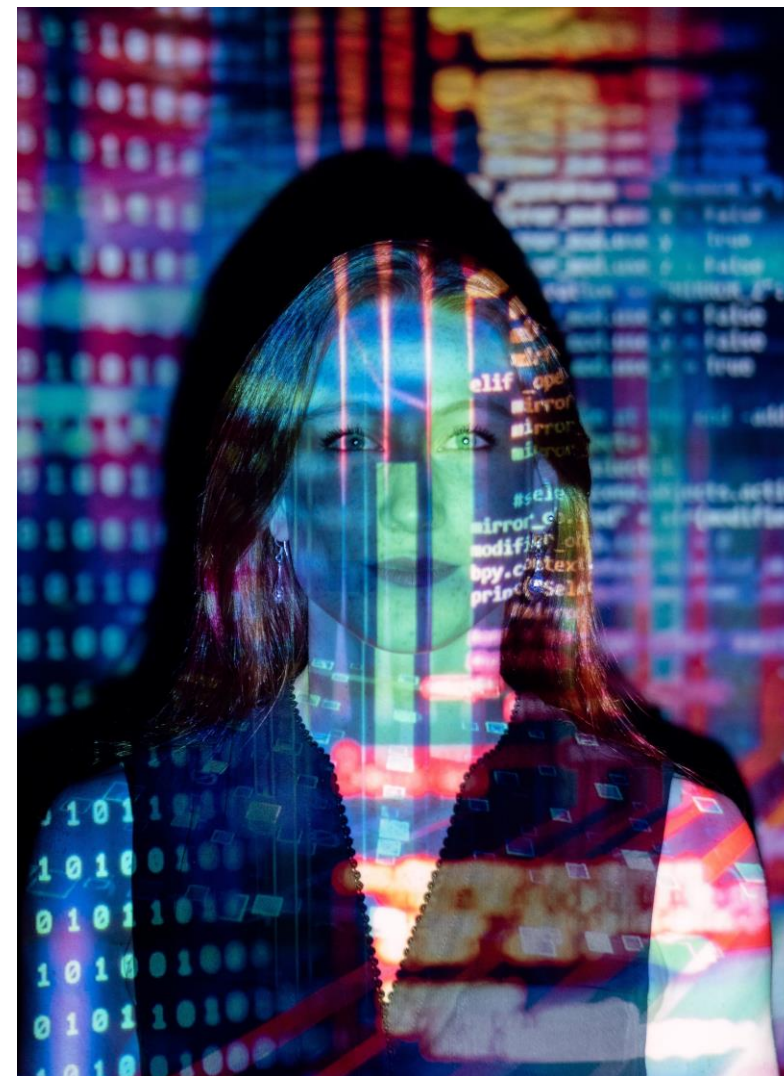
3ª semana



Simular requisições externas

Instalou o
Postman?

1. Criar uma nova Collection no **Postman** chamada Turma1.
2. Crie uma nova requisição em Add request chamada Cadastrar.
3. Escolha o método POST.
4. Na aba Body escolha raw e o formato JSON.

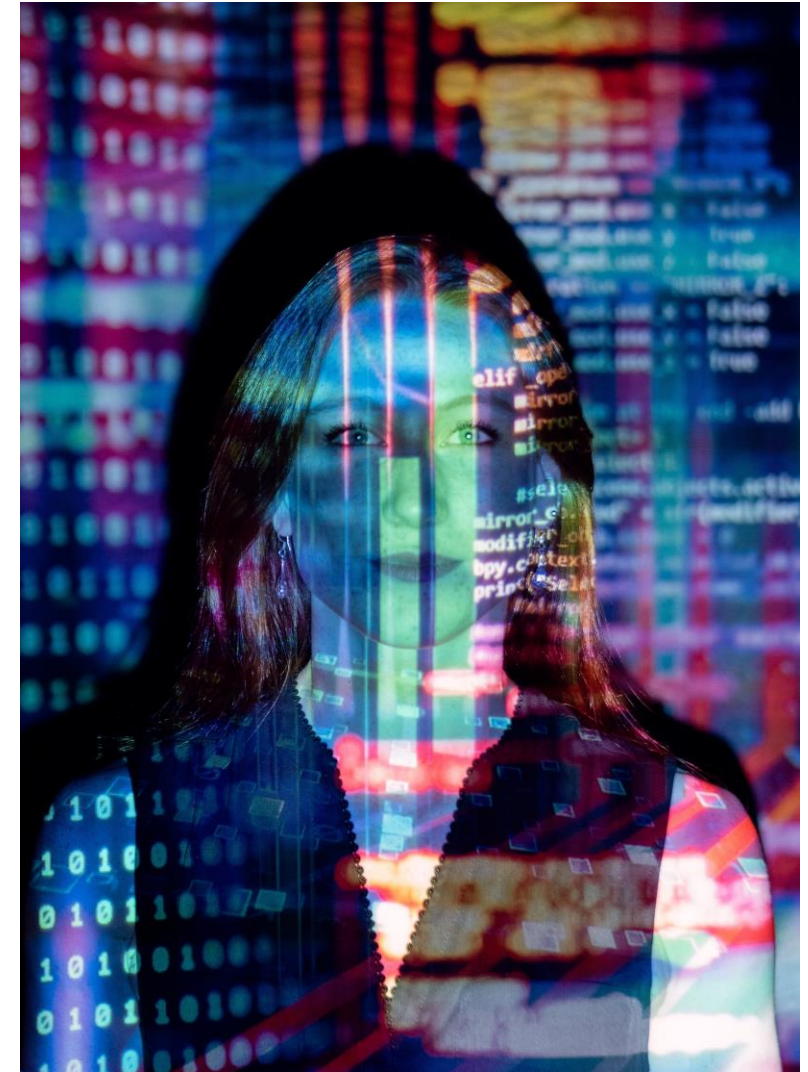


Simular requisições externas

5. Altere o método `get` para `post`.

```
13 app.post('/servicos', async (req, res) => {  
14   let create=await servico.create({  
15     nome: "HTML/CSS",  
16     descricao: "Páginas estáticas estilizadas",  
17     createAt: new Date(),  
18     updateAt: new Date()  
19   });  
20   res.send('Serviço criado com sucesso!');  
21 });
```

6. Copie o endereço que você estava cadastrando pelo navegador para a requisição Cadastrar da collection Turma1 no Postman, `http://localhost:3000/servicos`, para cadastrar um novo serviço.

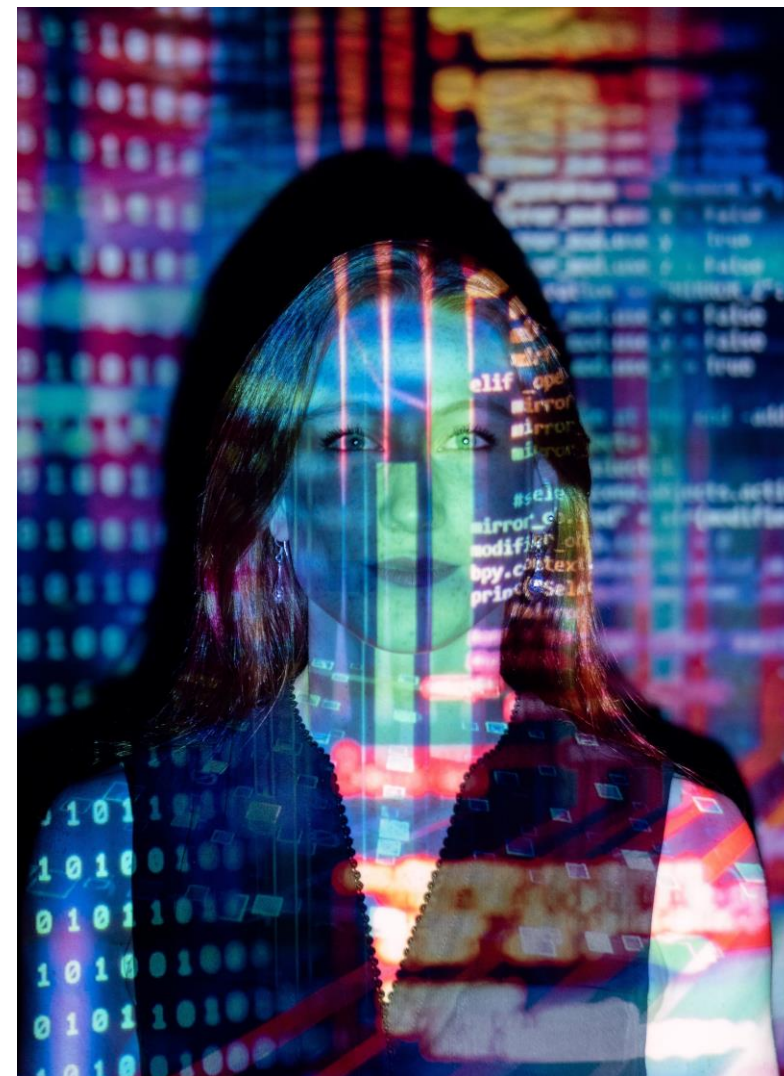


Simular requisições externas

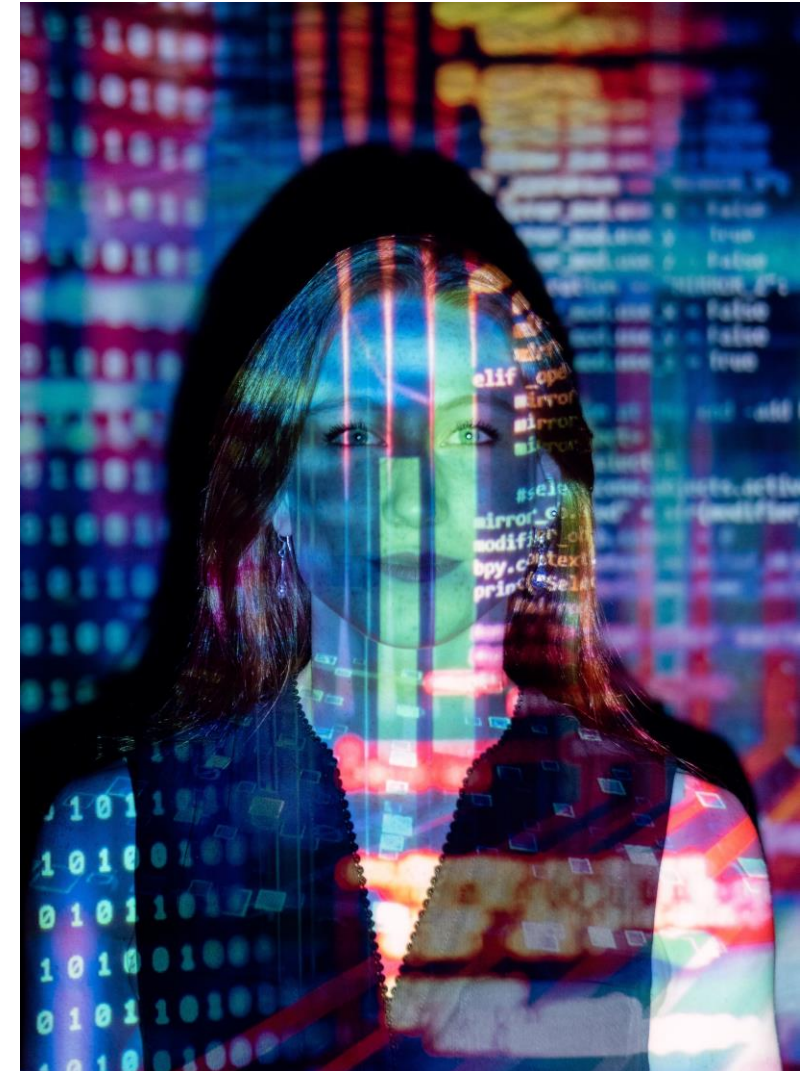
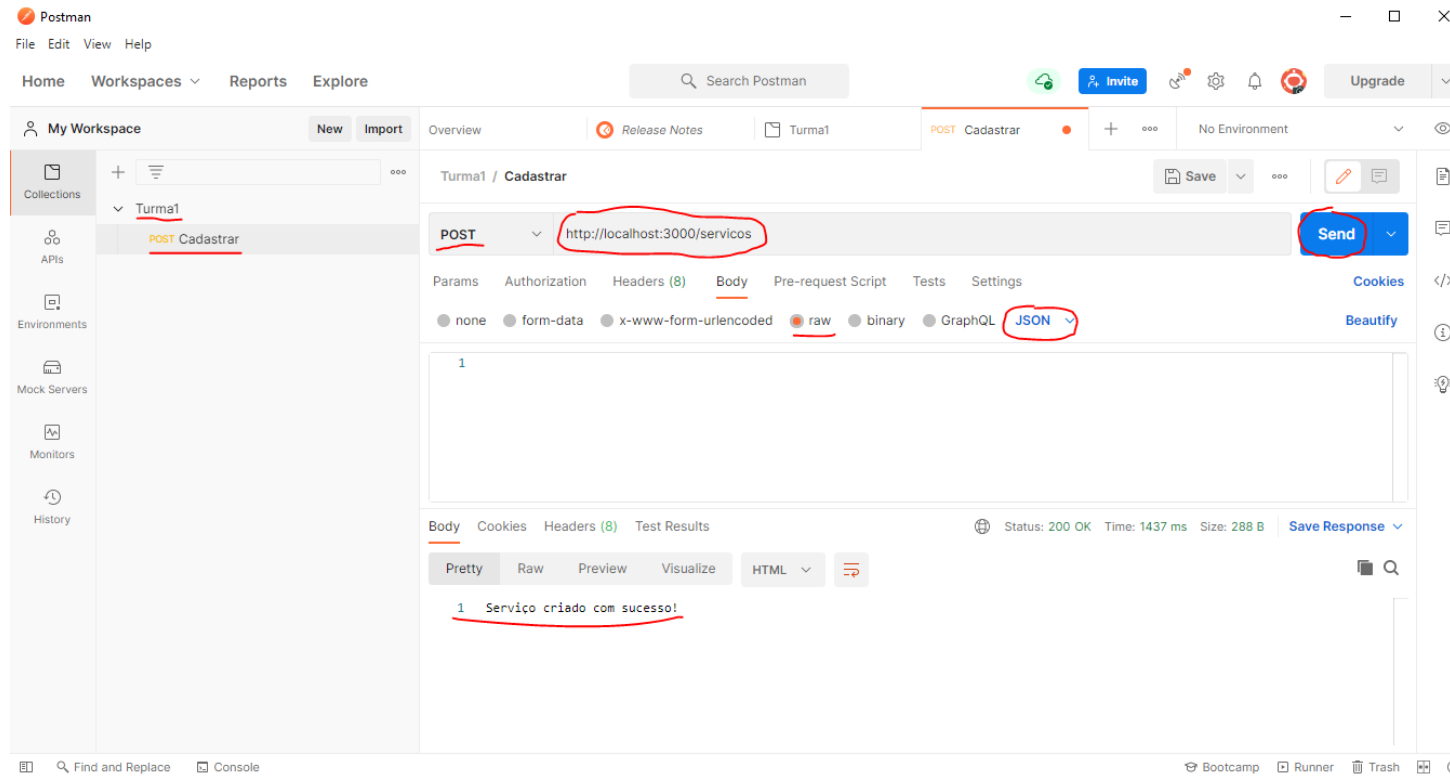
7. Altere estaticamente um novo registro como segue:

```
13 app.post('/servicos', async (req, res) => {  
14   let create=await servico.create({  
15     nome: "Delphi",  
16     descricao: "Manutenção e suporte a sistemas legados em Delphi",  
17     createAt: new Date(),  
18     updateAt: new Date()  
19   });  
20   res.send('Serviço criado com sucesso!');  
21 });
```

8. Execute no Postman, no botão Send.

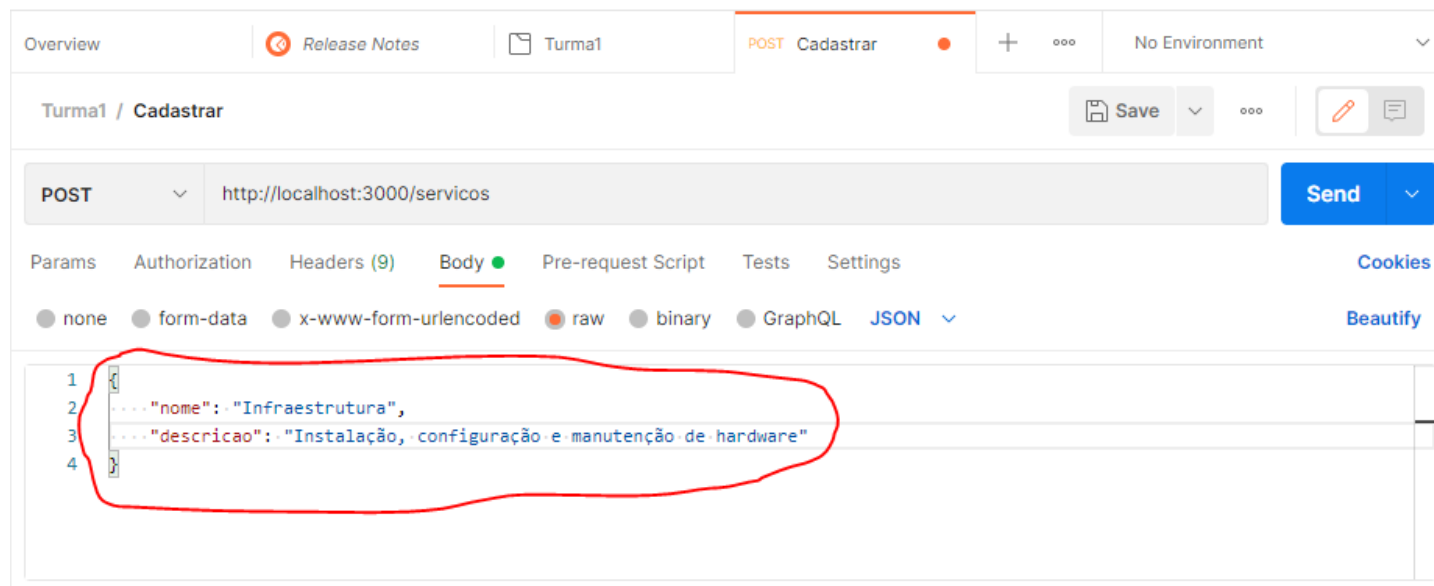


Simular requisições externas



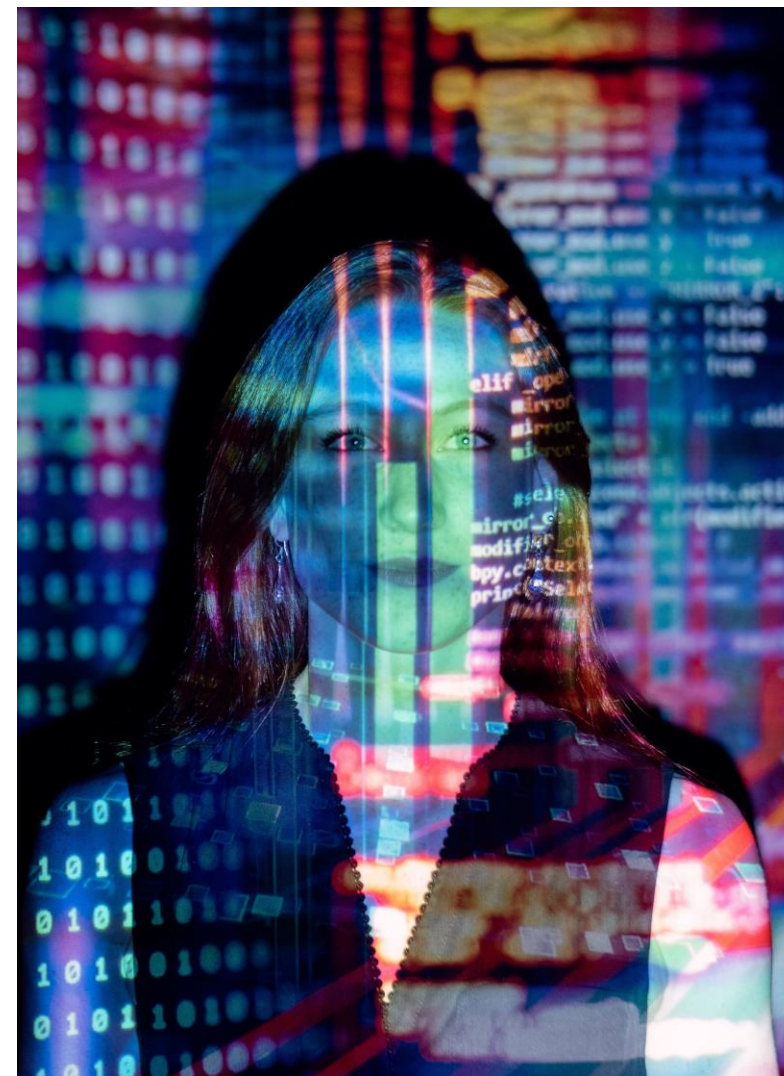
Simular requisições externas

9. Recorte as informações de cadastro no código e as insira no Postman, no formato JSON.



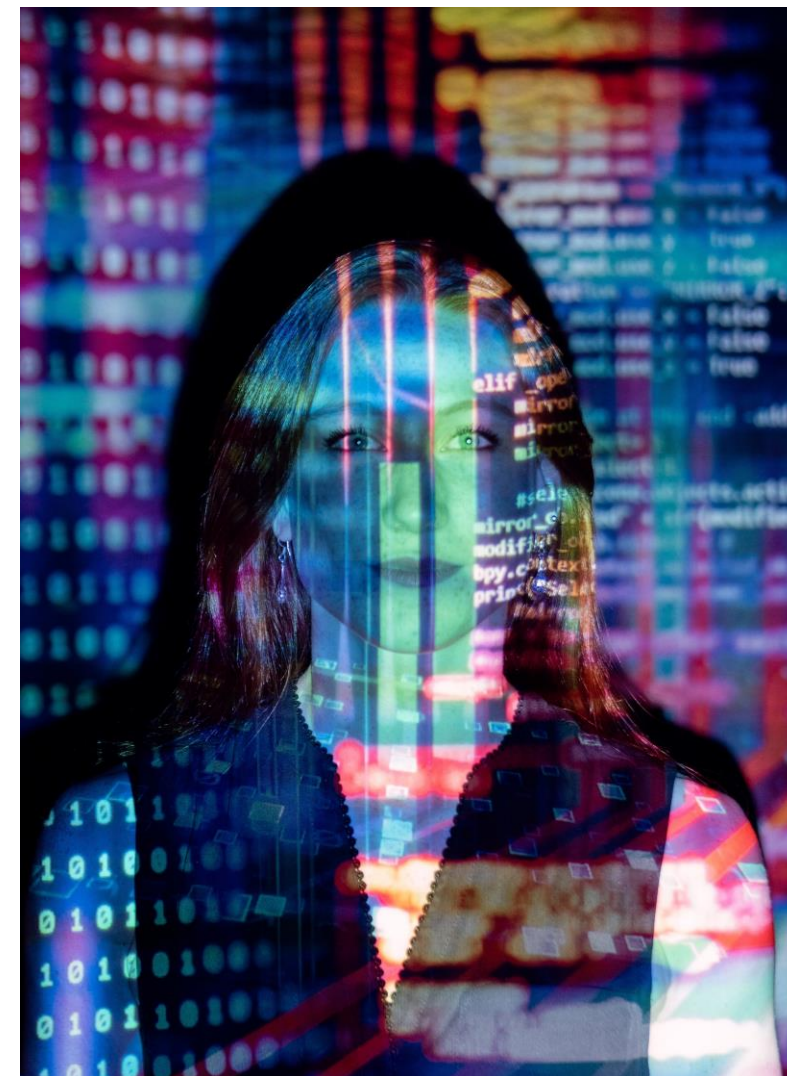
10. Informar na aplicação que ele utiliza o formato JSON.

11. Informar que os dados serão passados no corpo da requisição.



Simular requisições externas

```
JS Controller.js > ...
1  const express=require('express');
2  const cors=require('cors');
3
4  const models=require('./models');
5
6  const app=express();
7  app.use(cors());
8  app.use(express.json());
9
10 let cliente=models.Cliente;
11 let servico=models.Servico;
12 let pedido=models.Pedido;
13
14 app.post('/servicos', async (req, res) => {
15   let create=await servico.create(
16     req.body
17   );
18   res.send('Serviço criado com sucesso!');
19 });
```



Simular requisições externas

12. Envie a requisição pelo POSTMAN.

13. Se o resultado for Serviço criado com sucesso, verifique no phpMyAdmin o registro cadastrado.

A mostrar registos de 0 - 3 (4 total, A consulta demorou 0,0003 segundos.)

SELECT * FROM `servicos`

☐Mostrar tudo

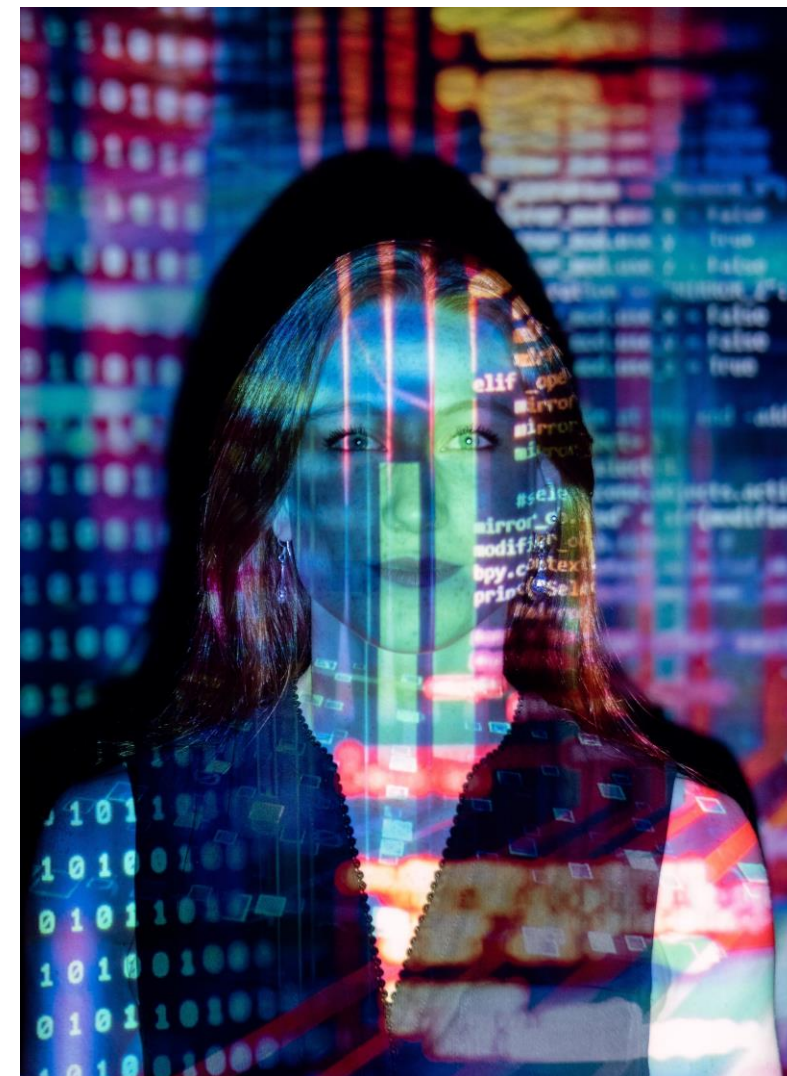
Número de registos: 25

Filtrar registos:

Pesquisar esta tabela

Ordenar pela chave: Nenhum

+ Opções

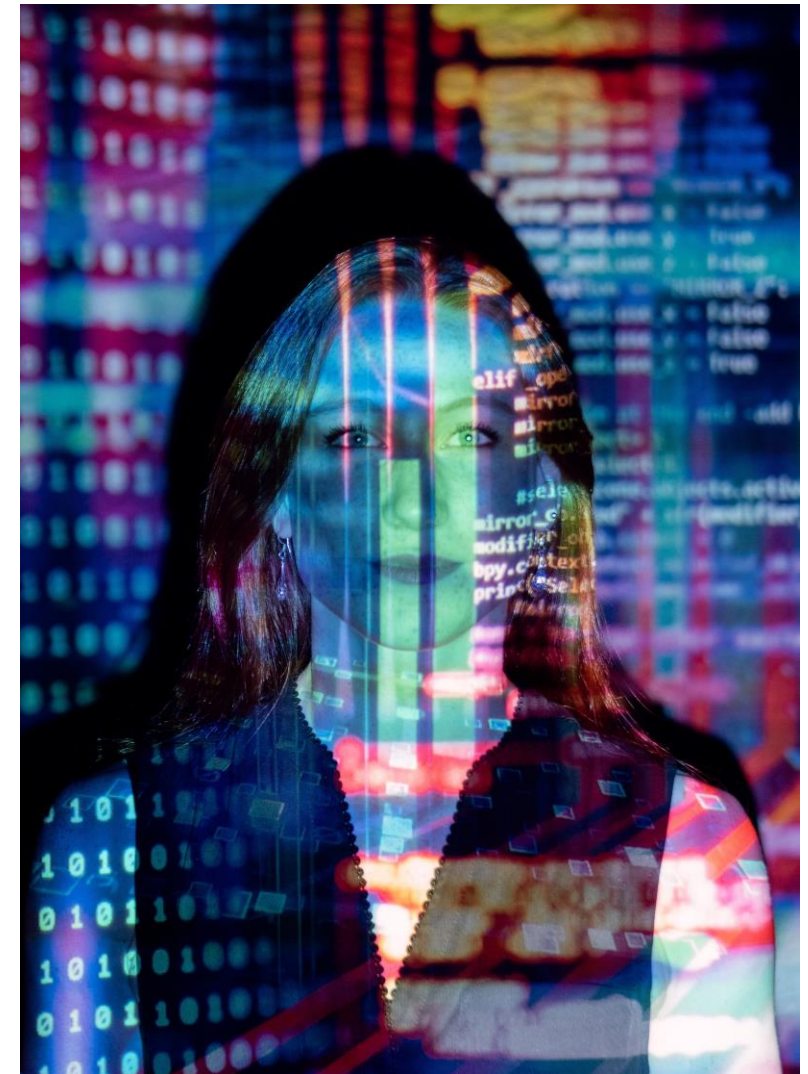


Verificar se foi cadastrado com sucesso

1. Altere o código para verificar se houve erro.

```
14 app.post('/servicos', async (req, res) => {
15   let create=await servico.create(
16     req.body
17   ).then(function (){
18     return res.json({
19       error: false,
20       message: "Serviço criado com sucesso!"
21     })
22   }).catch(function(erro){
23     return res.status(400).json({
24       error: true,
25       message: "Erro no cadastro do serviço."
26     })
27   });
28 });
```

2. Crie um novo serviço no Postman e execute a requisição



Verificar se foi cadastrado com sucesso

Overview | Release Notes | Turma1 | POST Cadastrar | + | No Environment

Turma1 / Cadastrar | Save |

POST | http://localhost:3000/servicos | Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies | Beautify

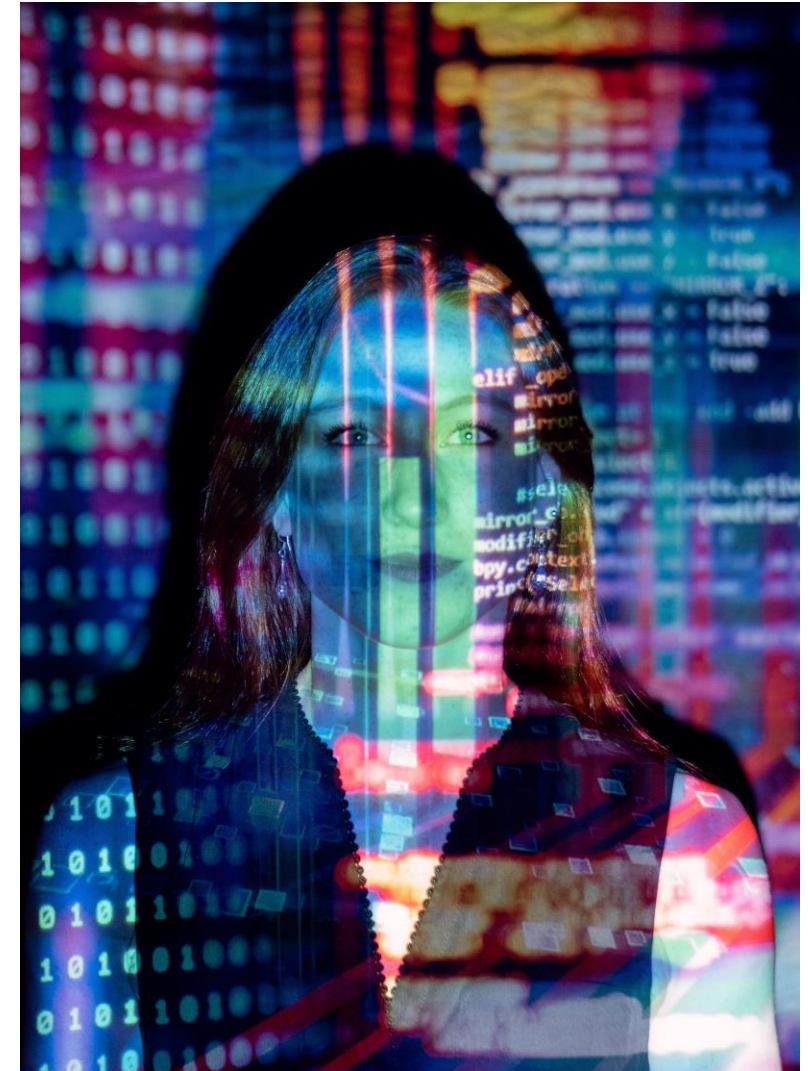
none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
1 {
2   ...."nome": "Rede de computadores",
3   ...."descricao": "Desenvolve projeto de redes locais e implementa a solução"
4 }
```

Body | Cookies | Headers (8) | Test Results | Status: 200 OK | Time: 184 ms | Size: 323 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2   "error": false,
3   "message": "Serviço criado com sucesso!"
4 }
```



Exercícios

Até aqui tudo bem?

Então agora é com você. O que você aprendeu hoje? Vamos testar?

Implemente o cadastro de clientes
(insira 5 novos clientes)

Implemente o cadastro de pedidos
(insira 10 novos pedidos)

Lembre-se que para cada pedido você deve inserir pelo menos 1 item.



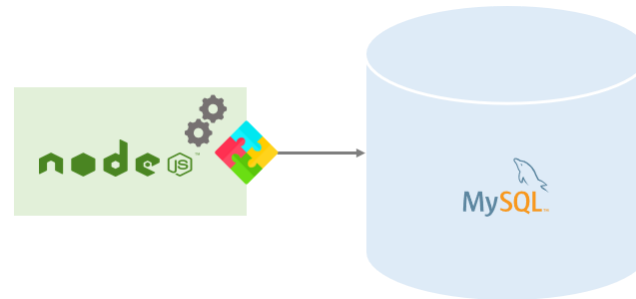
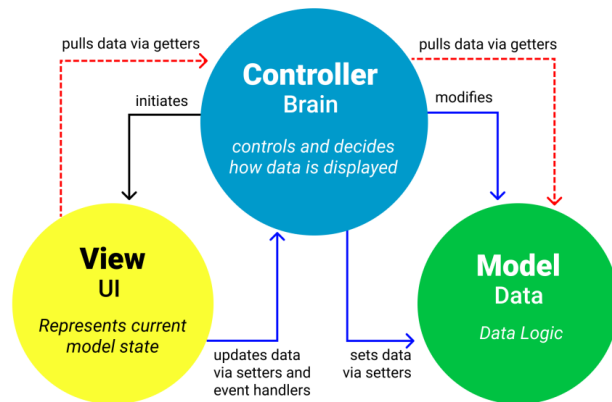
Não se esqueça de mim...



O que estudamos até aqui...

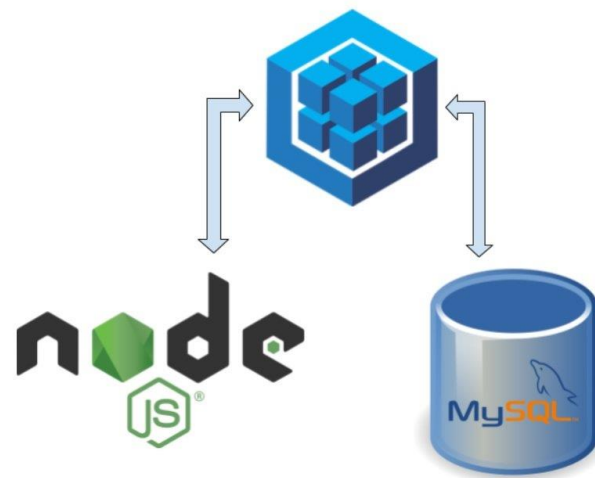
Temos a base de dados pronta.
Implementamos o método para cadastrar.

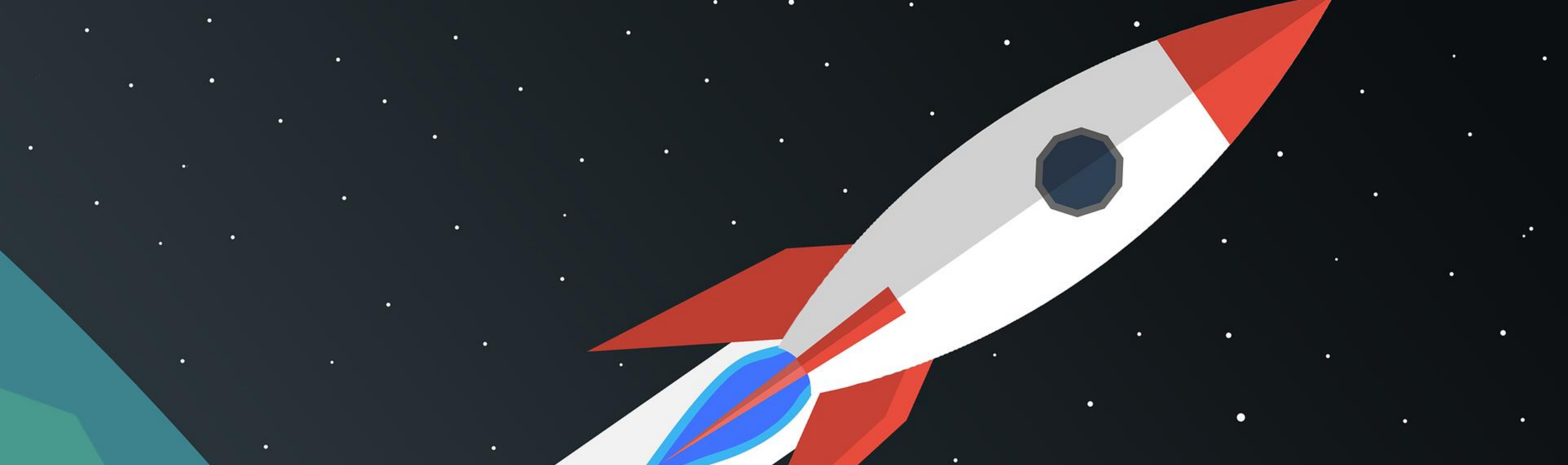
MVC Architecture Pattern



O que vem depois

Agora é a vez de implementar a consulta.





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 8 | Profº. Erinaldo



@tiacademybrasil

CRUD - Consulta

Vamos praticar?

Dia 2: CRUD - Inserção

1. Criação da camada de controle da aplicação.
2. Criação do método *create*
3. Simular requisições externas
4. Verificar se foi cadastrado com sucesso

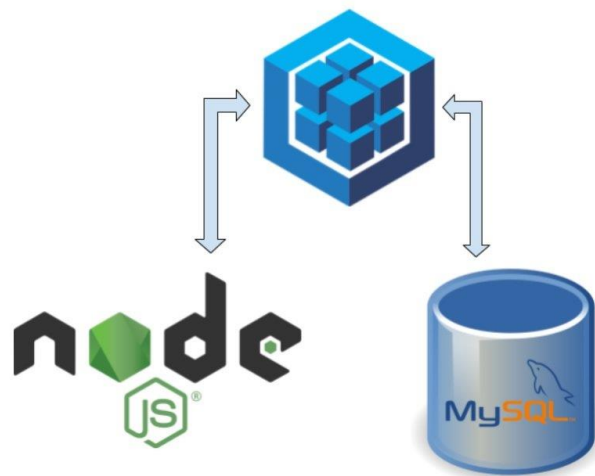
TOTAL: 22 horas

3ª semana



Dia 3: CRUD - Consulta

Criar uma rota de seleção



TOTAL: 22 horas

3ª semana

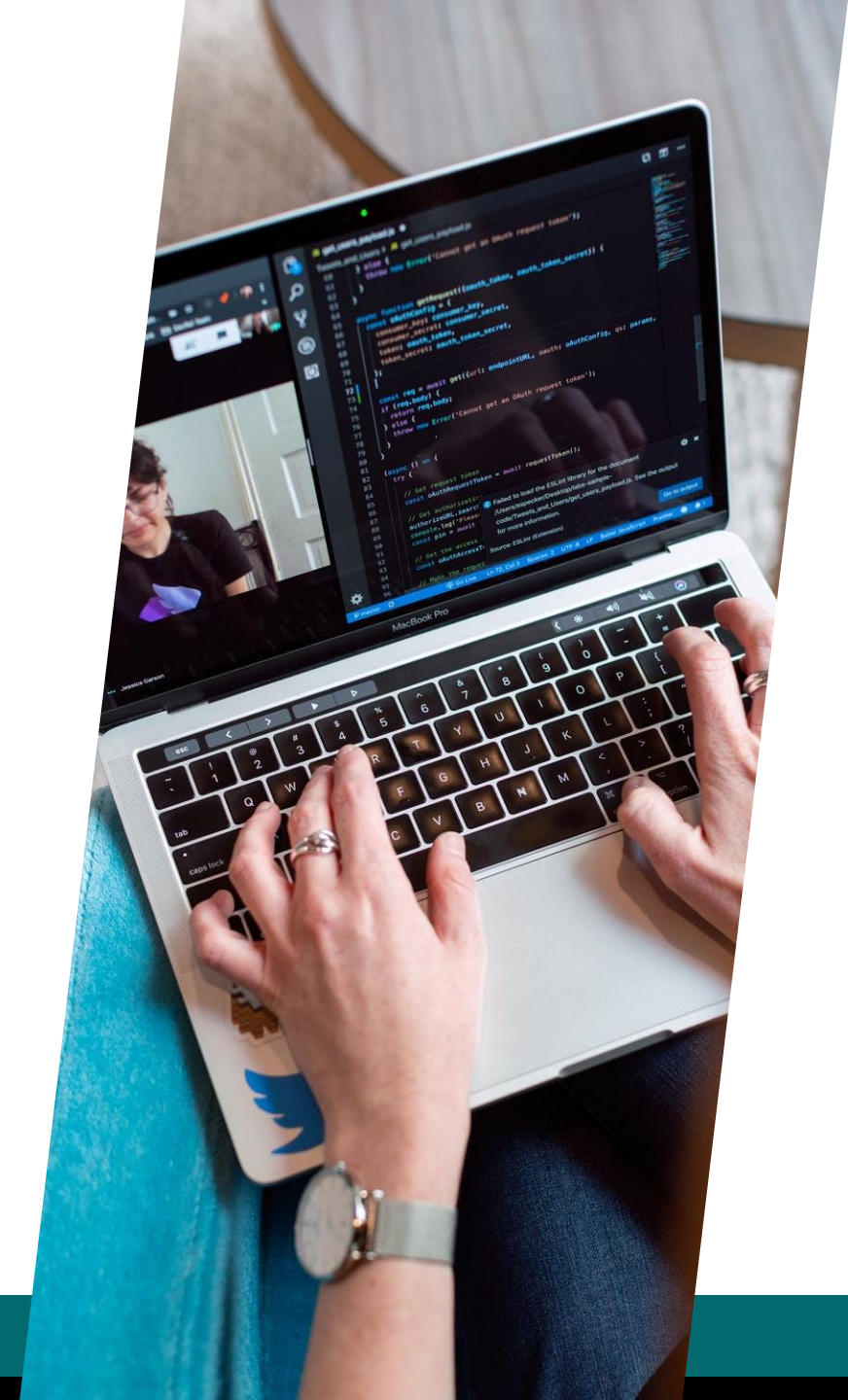


Antes de começar...

Você já criou um projeto utilizando o Nodejs, instalou dependências, criou o modelo de dados e exportou para o servidor de dados da aplicação.

Ontem você começou o CRUD da aplicação e implementou as inserções.

Hoje você vai dar continuidade ao CRUD implementando uma etapa muito importante na manipulação de dados, a consulta ou recuperação dos dados.



Criar uma rota de seleção

1. Abra o arquivo Controller.js e edite o código a seguir.

```
63  ✓ app.get('/listaservicos', async(req,res)=>{  
64  ✓    let read=await servico.findAll({  
65  ✓      raw:true  
66  ✓    }).then(function(servicos){  
67  ✓      res.json({servicos})  
68  ✓    });  
69  ✓  });
```

2. Execute a rota <http://localhost:3000/selecionaservicos>, o resultado deverá aparecer no terminal.



Criar uma rota de seleção

3. Adicione uma requisição no Postman para testar.

The screenshot shows the Postman interface with a workspace named 'My Workspace'. On the left sidebar, the 'Collections' tab is active, showing a collection named 'Turma1' with two requests: 'POST Cadastrar' and 'GET Listar'. The 'GET Listar' request is selected. The main panel shows the details of this request:

- Method:** GET
- URL:** http://localhost:3000/listaservicos
- Headers (7):** A table with one visible header:

KEY	VALUE
Key	Value
- Body:** The 'Body' tab is selected, showing a JSON response in 'Pretty' format:

```
1 {
2   "servicos": [
3     {
4       "id": 1,
5       "nome": "HTML/CSS",
6       "descricao": "Páginas estáticas estilizadas",
7       "createdAt": "2021-08-17T18:16:08.000Z",
8       "updatedAt": "2021-08-17T18:16:08.000Z"
9     }
10  ]
11 }
```
- Status:** 200 OK



O que estudamos até aqui...

Criar uma rota de seleção

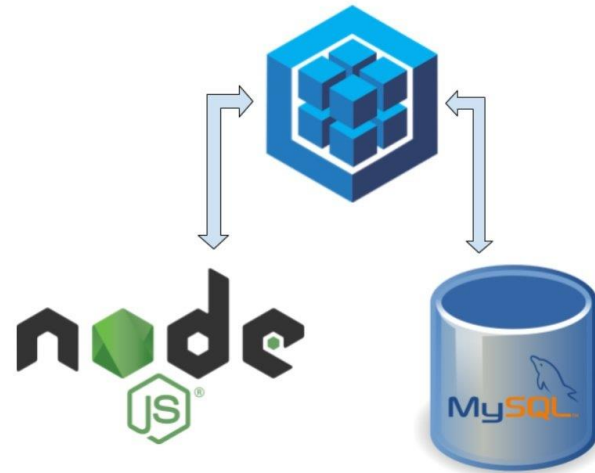


O que vem depois

Selecionar os dados em uma ordem

Retornar a quantidade de serviços

Visualizar os dados de um único serviço





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 9 | Profº. Erinaldo



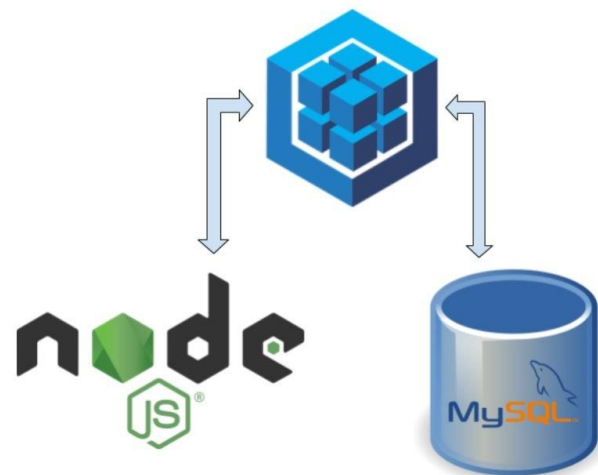
@tiacademybrasil

CRUD - Consulta

Vamos praticar?

Dia 3: CRUD - Consulta

Criar uma rota de seleção



TOTAL: 22 horas

3ª semana



Dia 3: CRUD - Consulta

1. Selecionar os dados em uma ordem
2. Retornar a quantidade de serviços
3. Visualizar os dados de um único serviço

TOTAL: 22 horas

3ª semana



Selecionar os dados em uma ordem

1. Selecione na documentação do Sequelize, <https://sequelize.org/master/manual/model-querying-basics.html#ordering-and-grouping>, a parte de ordenação e agrupamento.
2. Altere o código original para exibir a lista de serviços ordenada por nome.

```
63 app.get('/listaservicos', async(req,res)=>{
64   let read=await servico.findAll({
65     order: [['nome', 'DESC']]
66   }).then(function(servicos){
67     res.json({servicos})
68   });
69 });
```



Retornar a quantidade de serviços

1. Selecione na documentação do Sequelize, <https://sequelize.org/master/manual/model-querying-basics.html#code-count-code->, a parte de métodos de contagem.
2. Altere o código original para exibir a quantidade de serviços ofertados.

```
71 | app.get('/ofertas', async(req,res)=>{  
72 |   let read=await servico.count('id').then(function(servicos){  
73 |     res.json(servicos);  
74 |   });  
75 | });
```

Teste no Postman.



Visualizar os dados de um único serviço

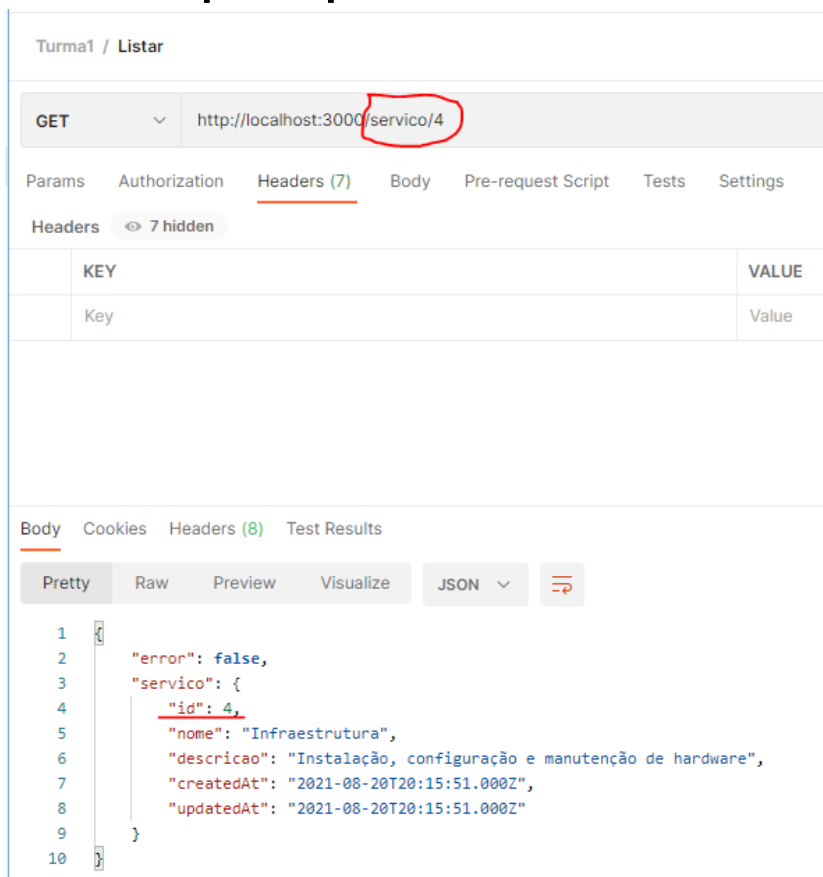
1. Crie uma nova rota para informar o id do serviço e visualizar o serviço caso este id esteja cadastrado.

```
77 app.get('/servico/:id', async(req, res) =>{
78   servico.findByPk(req.params.id)
79   .then(servico =>{
80     return res.json({
81       error: false,
82       servico
83     });
84   }).catch(function(erro){
85     return res.status(400).json({
86       error: true,
87       message: "Erro: Código não cadastrado!"
88     });
89   });
90 });
```



Visualizar os dados de um único serviço

2. Teste no Postman, informando a url e o id a ser pesquisado.

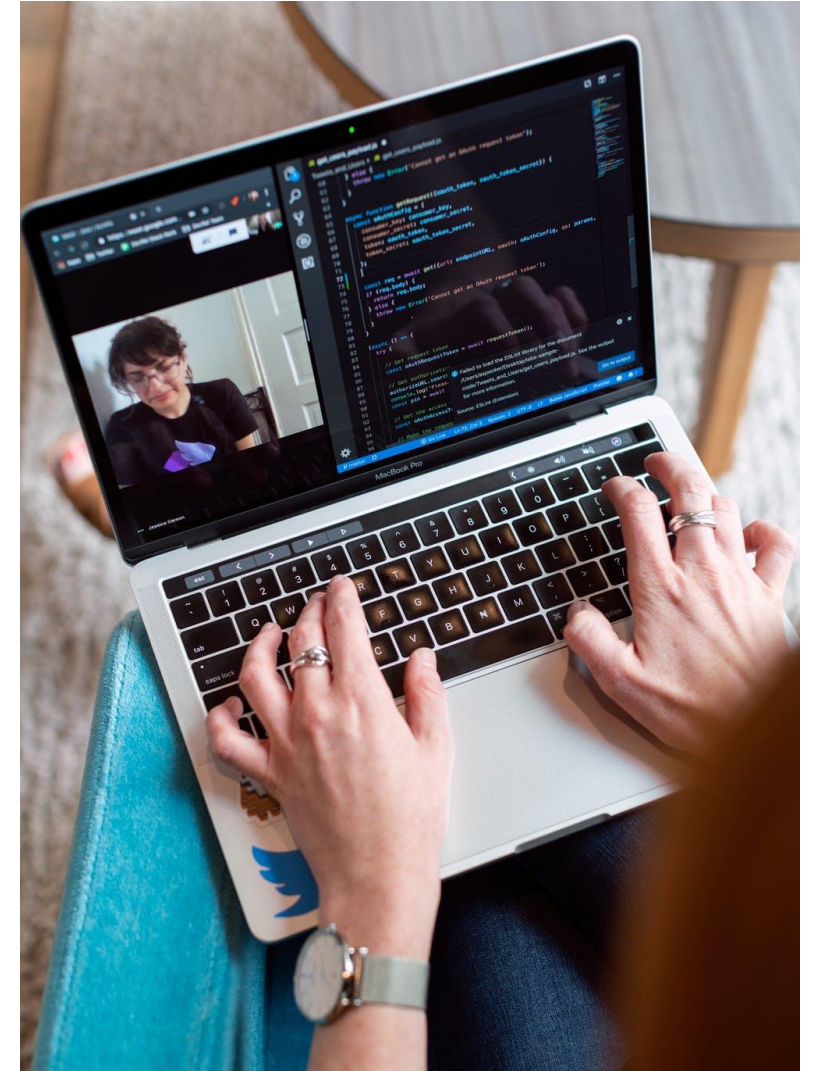


Teste outro id
não cadastrado



Exercícios

1. Visualize todos os clientes.
2. Visualize os clientes em ordem de antiguidade.
3. Visualize todos os pedidos.
4. Visualize o pedido por ordem a partir do maior valor.
5. Informe quantos clientes estão na nossa base de dados.
6. Informe a quantidade de pedidos solicitados.



O que estudamos até aqui...

Criar uma rota de seleção

Selecionar os dados em uma ordem

Retornar a quantidade de serviços

Visualizar os dados de um único serviço

O que vem depois

Temos a base de dados pronta.

Implementamos o método para cadastrar e a consulta.

Agora vamos aprender a excluir e alterar os registros cadastrados em nossa base de dados.





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 10 | Profº. Erinaldo



@tiacademybrasil

CRUD - Alteração

Vamos praticar?

Dia 3: CRUD - Consulta

1. Criar uma rota de seleção
2. Selecionar os dados em uma ordem
3. Retornar a quantidade de serviços
4. Visualizar os dados de um único serviço

TOTAL: 22 horas

3ª semana



Dia 4: CRUD – Alteração

1. Criar uma rota para fazer *update*



TOTAL: 22 horas

3ª semana

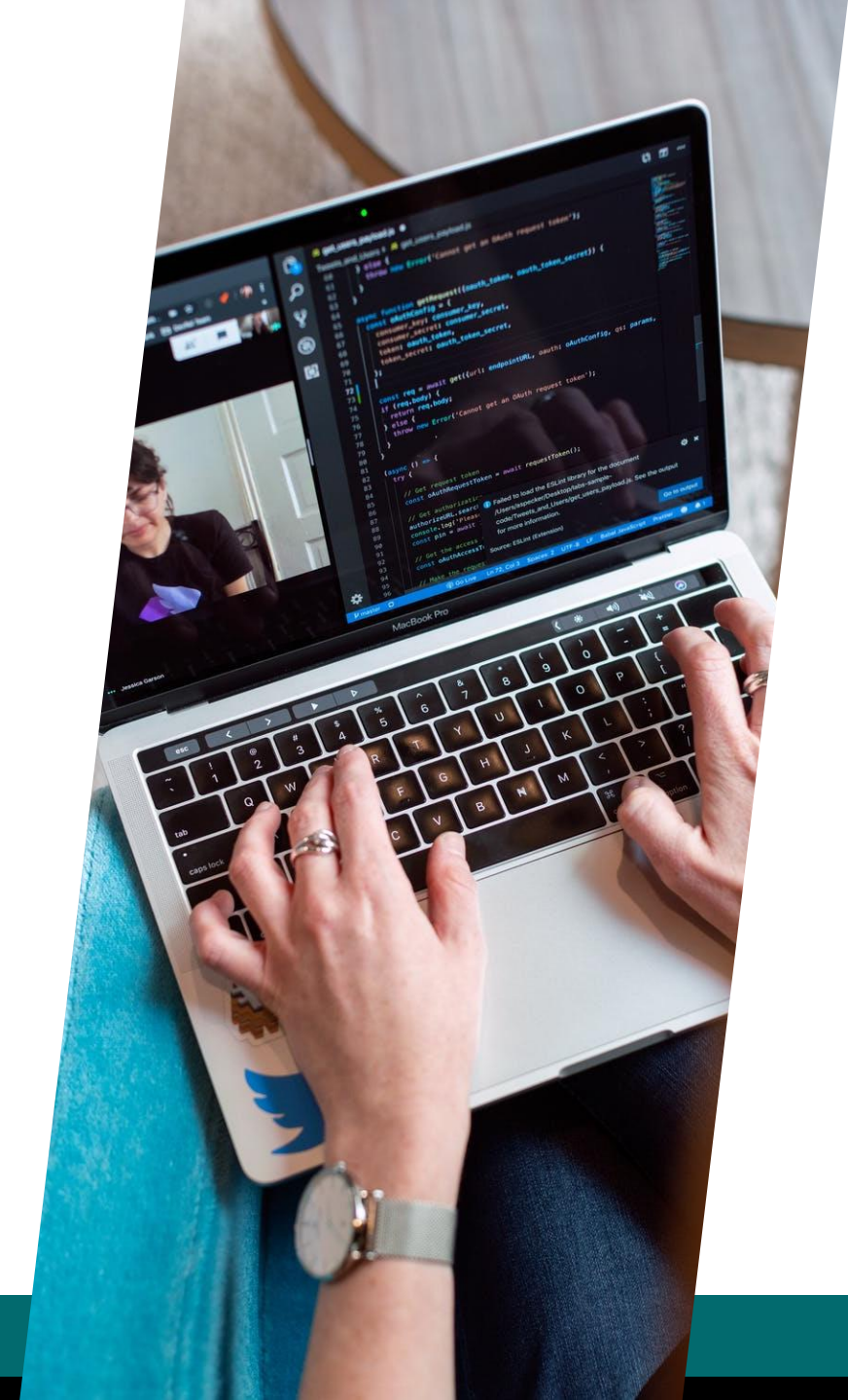


Antes de começar...

Você já criou um projeto utilizando o Nodejs, instalou dependências, criou o modelo de dados e exportou para o servidor de dados da aplicação.

Na sequência começou o CRUD da aplicação. Implementou as inserções e a consulta ou recuperação dos dados.

Na aula de hoje você vai implementar duas funcionalidades importantes na aplicação, a alteração e exclusão dos dados. Vamos começar!



Criar uma rota para fazer update

1. Faça uma consulta por id de serviço.

```
105 app.get('/atualizaseruico', async(req, res)=>{  
106     await servico.findByPk(1)  
107     .then(servico =>{  
108         return res.json({servico});  
109     });  
110 });
```

2. Execute o comando no navegador.
3. Em seguida informe os dados a serem atualizados.
4. Execute novamente no navegador.
5. Confirme a mudança no banco de dados.



■ Criar uma rota para fazer update

```
105 app.get('/atualizaseruico', async(req, res)=>{
106     await seruico.findByPk(1)
107     .then(seruico =>{
108         seruico.nome='HTML/CSS/JS';
109         seruico.descricao='P ginas est ticas e din micas estilizadas';
110         seruico.save();
111         return res.json({seruico});
112     });
113 });
```


O que estudamos até aqui...

Criar uma rota para fazer *update*

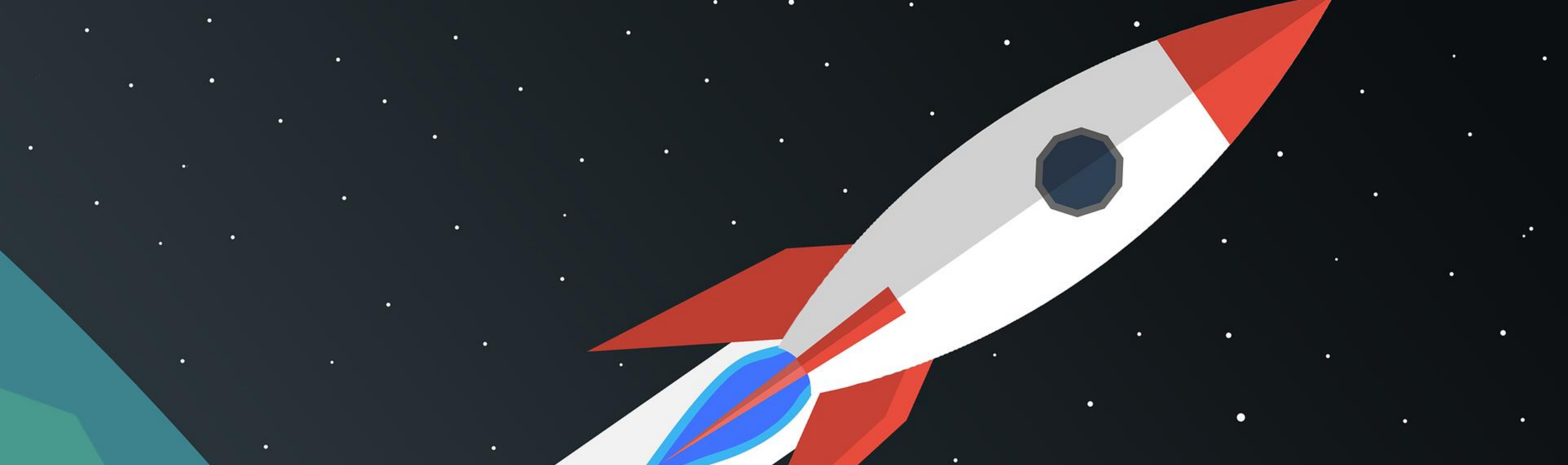


O que vem depois

Editar um registro com *put*

Combinar consulta e edição





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 11 | Profº. Erinaldo



@tiacademybrasil

CRUD - Alteração

Vamos praticar?

Dia 4: CRUD – Alteração

1. Criar uma rota para fazer *update*



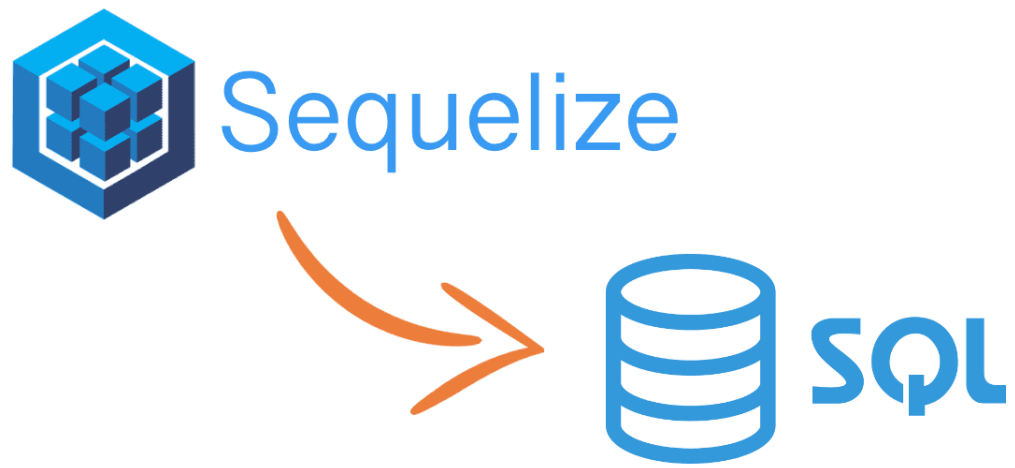
TOTAL: 22 horas

3ª semana



Dia 4: CRUD – Alteração

Editar um registro com *put*



TOTAL: 22 horas

3ª semana



Editar um registro com *put*

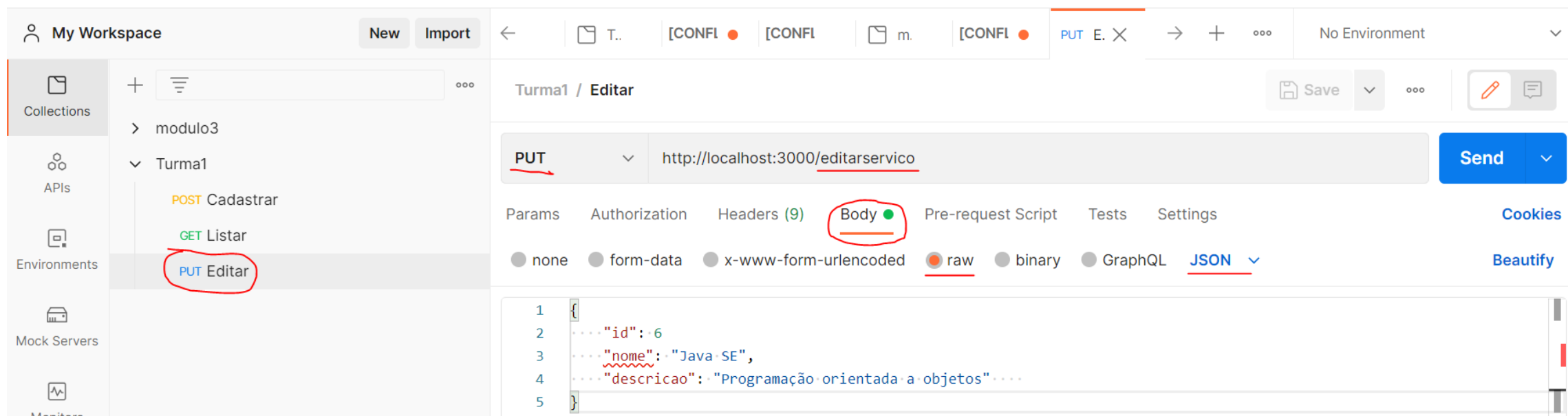
1. Crie uma nova rota para editar um registro da tabela serviços pelo corpo (*body*) da requisição.

```
115 app.put('/editarservico', (req, res)=>{
116     servico.update(req.body,{
117         where: {id: req.body.id}
118     }).then(function(){
119         return res.json({
120             error: false,
121             message: "Serviço modificado com sucesso!"
122         });
123     }).catch(function(erro){
124         return res.status(400).json({
125             error: true,
126             message: "Erro na modificação do serviço!"
127         });
128     });
129 });
```



Editar um registro com *put*

2. Faça uma nova requisição no Postman.



Editar um registro com *put*

Turma1 / Editar

PUT http://localhost:3000/editarservico **Send**

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON** Beautify

```
1 {
2   "id": 6,
3   "nome": "Java SE",
4   "descricao": "Programação orientada a objetos"
5 }
```

Body Cookies Headers (8) Test Results 200 OK 5.89 s 327 B Save Response

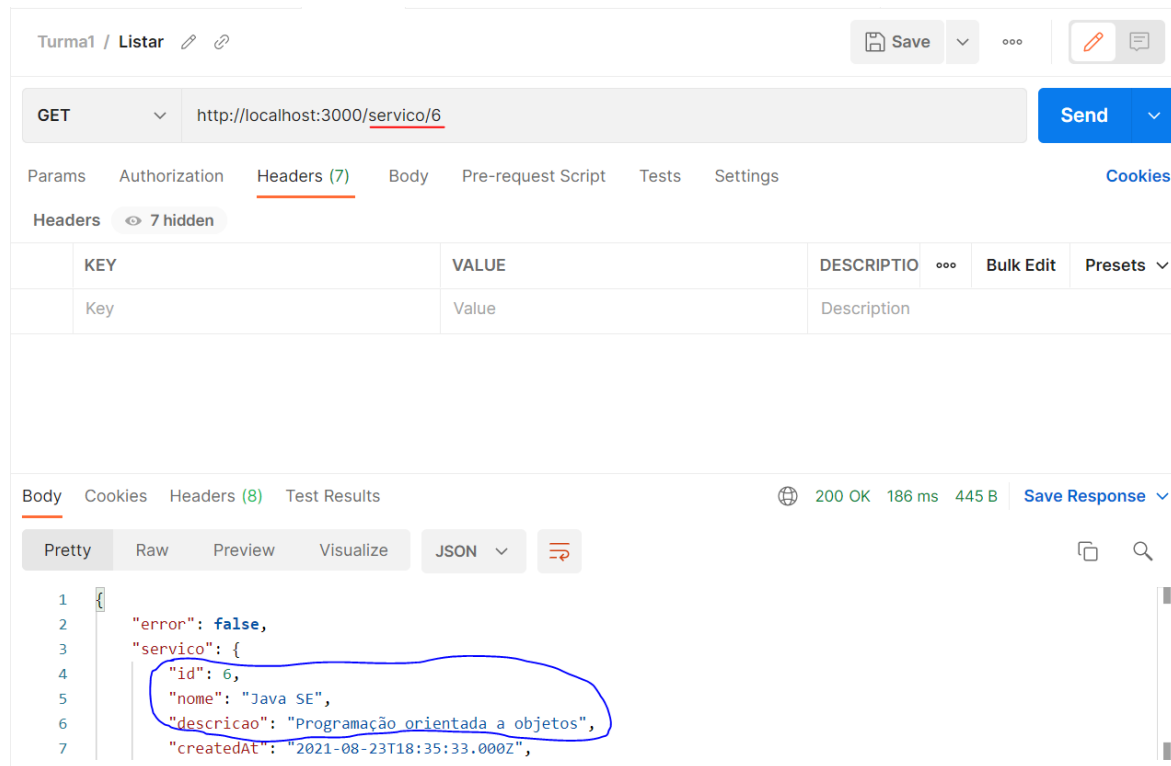
Pretty Raw Preview Visualize **JSON**

```
1 {
2   "error": false,
3   "message": "Serviço modificado com sucesso!"
4 }
```



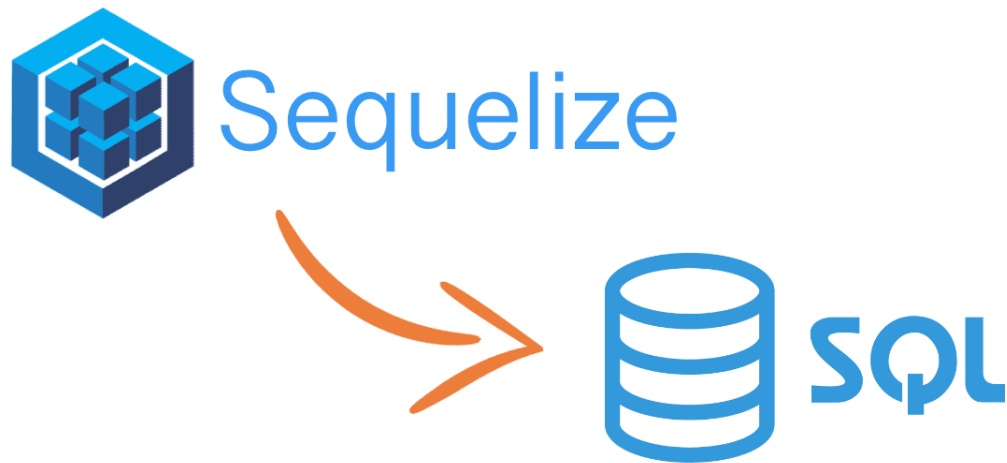
Editar um registro com *put*

- Para verificar se realmente a alteração foi feita, utilize no Postman, a rota `/servico/:id`, informando o `id` alterado.



O que estudamos até aqui...

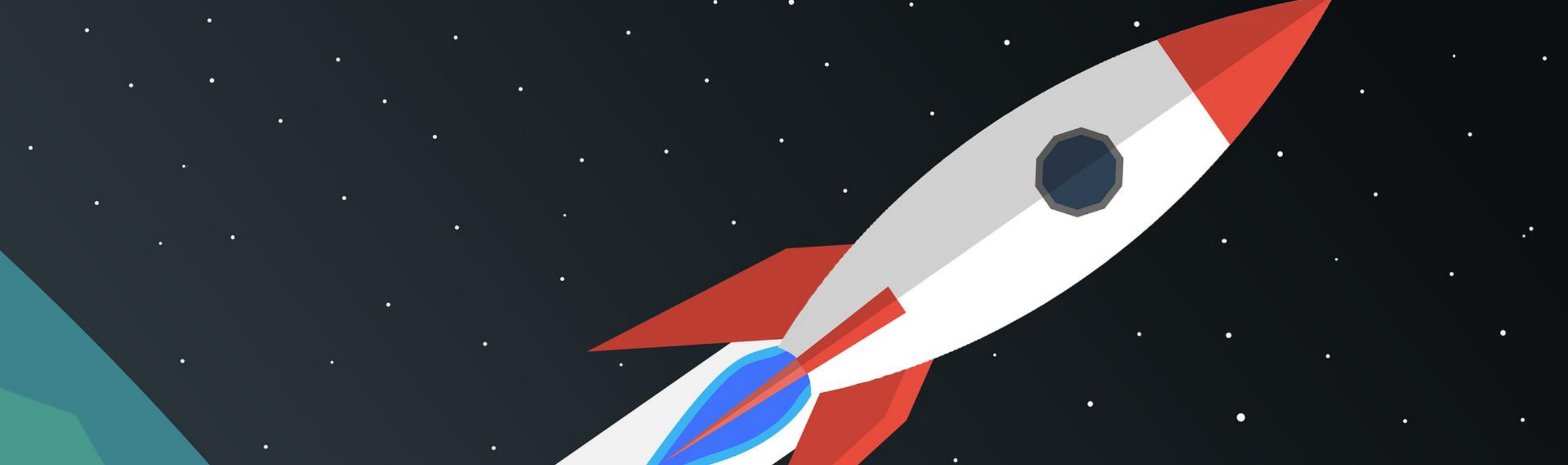
1. Editar um registro com *put*



O que vem depois

Combinar consulta e edição





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 12 | Profº. Erinaldo



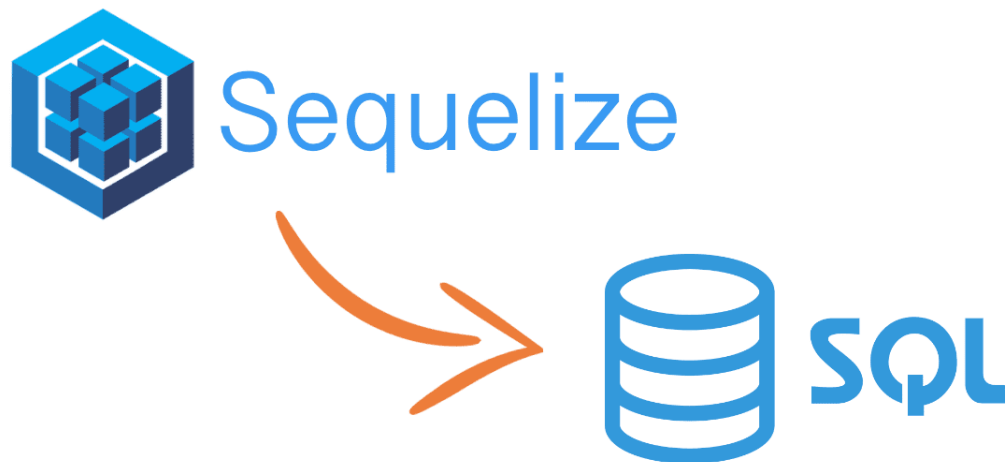
@tiacademybrasil

CRUD - Alteração

Vamos praticar?

Dia 4: CRUD – Alteração

Editar um registro com *put*



TOTAL: 22 horas

3ª semana



Dia 4: CRUD – Alteração

Combinar consulta e edição



TOTAL: 22 horas

3ª semana



Ajustar as associações

1. Associação de Clientes e Pedidos.

```
12 static associate(models) {  
13   // define association here  
14   Cliente.hasMany(models.Pedido, {foreignKey: 'ClienteId', as: 'pedidos'});  
15 }  
16 };  
17 Cliente.init({
```

2. Associação entre Pedidos e Cliente, e associação entre Pedidos e Serviços.

```
12 static associate(models) {  
13   // define association here  
14   Pedido.belongsTo(models.Cliente, {foreignKey: 'ClienteId', as: 'clientes'});  
15   Pedido.belongsToMany(models.Servico, {  
16     foreignKey: 'ServicoId',  
17     through: 'ItemPedido', as: 'servicos_ped'  
18   });  
19   Pedido.hasMany(models.ItemPedido, {foreignKey: 'PedidoId', as: 'item_pedidos'});  
20 }  
21 };  
22 Pedido.init({
```



Ajustar as associações

3. Associação de Serviços e Pedidos.

```
12 static associate(models) {  
13   // define association here  
14   Servico.belongsToMany(models.Pedido, {  
15     foreignKey: 'PedidoId',  
16     through: 'ItemPedido', as: 'serv'  
17   });  
18   Servico.hasMany(models.ItemPedido, {foreignKey: 'ServicoId', as: 'item_servicos'});  
19 }  
20 };  
21 Servico.init({
```

4. Classe de associação ItemPedidos entre Pedidos e Serviços.

```
12 static associate(models) {  
13   // define association here  
14   ItemPedido.belongsTo(models.Pedido, {foreignKey: 'PedidoId', as: 'pedido'});  
15   ItemPedido.belongsTo(models.Servico, {foreignKey: 'ServicoId', as: 'servico'});  
16 }  
17 };  
18 ItemPedido.init({
```



Combinar consulta e edição

1. Para mostrar tudo relacionado a um pedido passado como parâmetro na requisição.

```
145 app.get('/pedidos/:id', async(req, res) => {  
146   await pedido.findByPk(req.params.id, {include: [{all: true}]})  
147   .then(ped => {  
148     return res.json({ped});  
149   });  
150 });
```

2. Teste o resultado dessa consulta no Postman.

Agora vamos lá!



Combinar consulta e edição

3. Implementar o editar pedido com base no servicold.

a) Verifique se o Sequelize foi importado.

```
1  const express = require('express');
2  const cors = require('cors');
3
4  const { Sequelize } = require('./models');
5
6  const models=require('./models');
```

b) Implemente o método *put* para alterar um item.



Combinar consulta e edição

```
154 app.put('/pedidos/:id/editaritem', async(req, res)=>{
155   const item = {
156     quantidade: req.body.quantidade,
157     valor: req.body.valor
158   }
159
160   if (!await pedido.findByPk(req.params.id)){
161     return res.status(400).json({
162       error: true,
163       message: 'Pedido não encontrado.'
164     });
165   };
166
167   if (!await servico.findByPk(req.body.ServicoId)){
168     return res.status(400).json({
169       error: true,
170       message: 'Serviço não encontrado.'
171     });
172   };
173
174   await itempedido.update(item,{
175     where: Sequelize.and({ServicoId: req.body.ServicoId},
176                          {PedidoId: req.params.id})
177   }).then(function(itens){
178     return res.json({
179       error: false,
180       message: "Pedido alterado com sucesso!",
181       itens
182     });
183   }).catch(function(erro){
184     return res.status(400).json({
185       error: true,
186       message: "Erro: não foi possível aterar."
187     });
188   });
189 });
```

O que será
alterado?

Verificar se o
pedido existe.

Verificar se o
serviço existe

Alterar o item
pedido.



Combinar consulta e edição

O que será alterado?

```
154 app.put('/pedidos/:id/editaritem', async(req, res)=>{
155   const item = {
156     quantidade: req.body.quantidade,
157     valor: req.body.valor
158   }
159 }
```



Combinar consulta e edição

Verificar se o pedido existe.

```
160     if (!await pedido.findByPk(req.params.id)){  
161         return res.status(400).json({  
162             error: true,  
163             message: 'Pedido não encontrado.'  
164         });  
165     };  
166
```



Combinar consulta e edição

Verificar se o serviço existe.

```
167     if (!await servico.findByPk(req.body.ServicoId)){  
168         return res.status(400).json({  
169             error: true,  
170             message: 'Serviço não encontrado.'  
171         });  
172     };  
173
```



Combinar consulta e edição

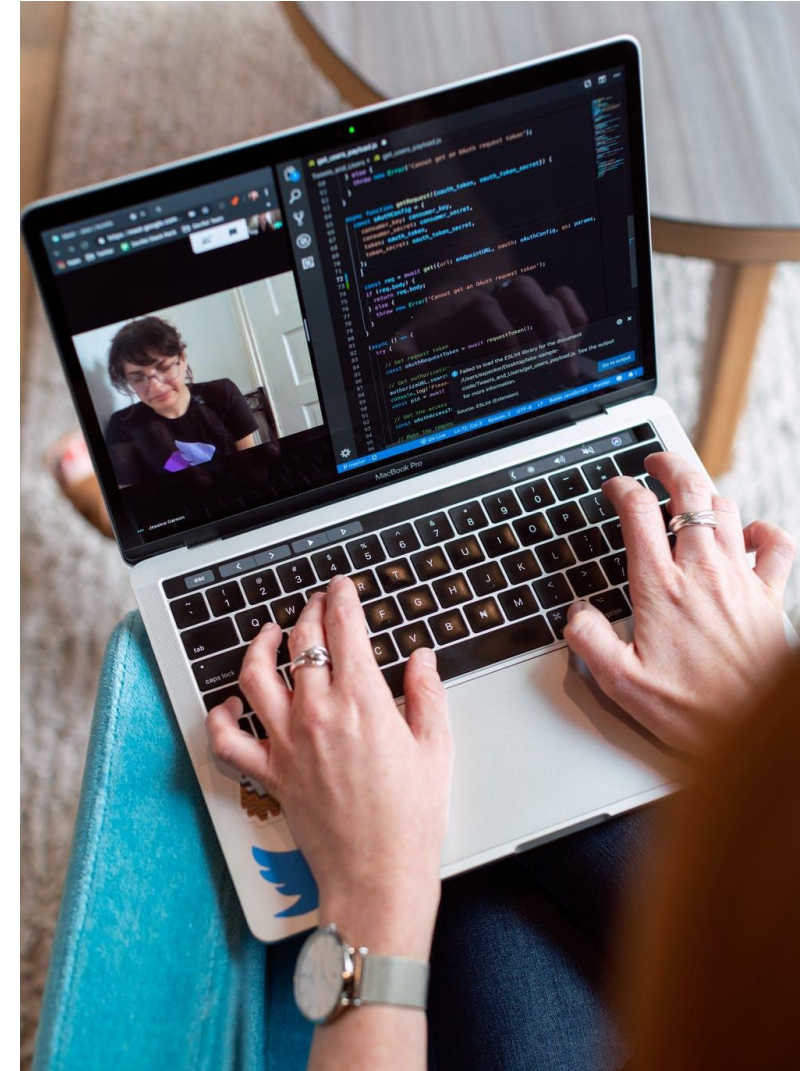
Alterar o item pedido.

```
174     await itempedido.update(item,{
175         where: Sequelize.and({ServicoId: req.body.ServicoId},
176                               {PedidoId: req.params.id})
177     }).then(function(itens){
178         return res.json({
179             error: false,
180             message: "Pedido alterado com sucesso!",
181             itens
182         });
183     }).catch(function(erro){
184         return res.status(400).json({
185             error: true,
186             message: "Erro: não foi possível alterar."
187         });
188     });
189 });
190
```



Exercícios

1. Faça a busca por serviços de clientes passando o id do cliente no corpo da requisição.
2. Utilize a rota para consultar clientes e faça a edição de um cliente pelo método *put*.
3. Utilize a rota para consultar pedidos e faça a edição de um pedido pelo método *put*.
4. Faça uma rota que liste todos os pedidos de um cliente.
5. Crie uma rota que permita alterar esse pedido utilizando o clienteld.



O que estudamos até aqui...

Criar uma rota para fazer *update*

Editar um registro com *put*

Combinar consulta e edição

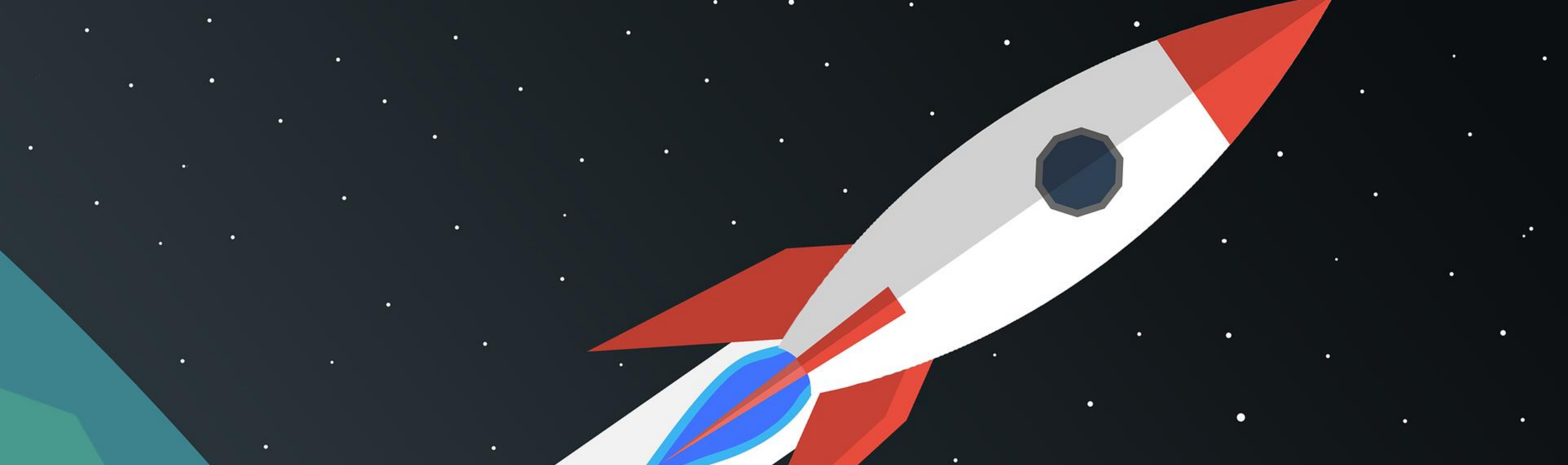


O que vem depois

Criar uma rota para fazer a exclusão

Excluir a partir de uma aplicação externa





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br



@tiacademybrasil

Back-end com Node.js

Aula 13 | Profº. Erinaldo



@tiacademybrasil

CRUD - Exclusão

Vamos praticar?

Dia 5: CRUD –Exclusão

1. Criar uma rota para fazer update
2. Editar um registro com put
3. Combinar consulta e edição

TOTAL: 22 horas

3ª semana



Dia 5: CRUD – Exclusão

1. Criar uma rota para fazer a exclusão
2. Excluir a partir de uma aplicação externa

TOTAL: 22 horas

3ª semana

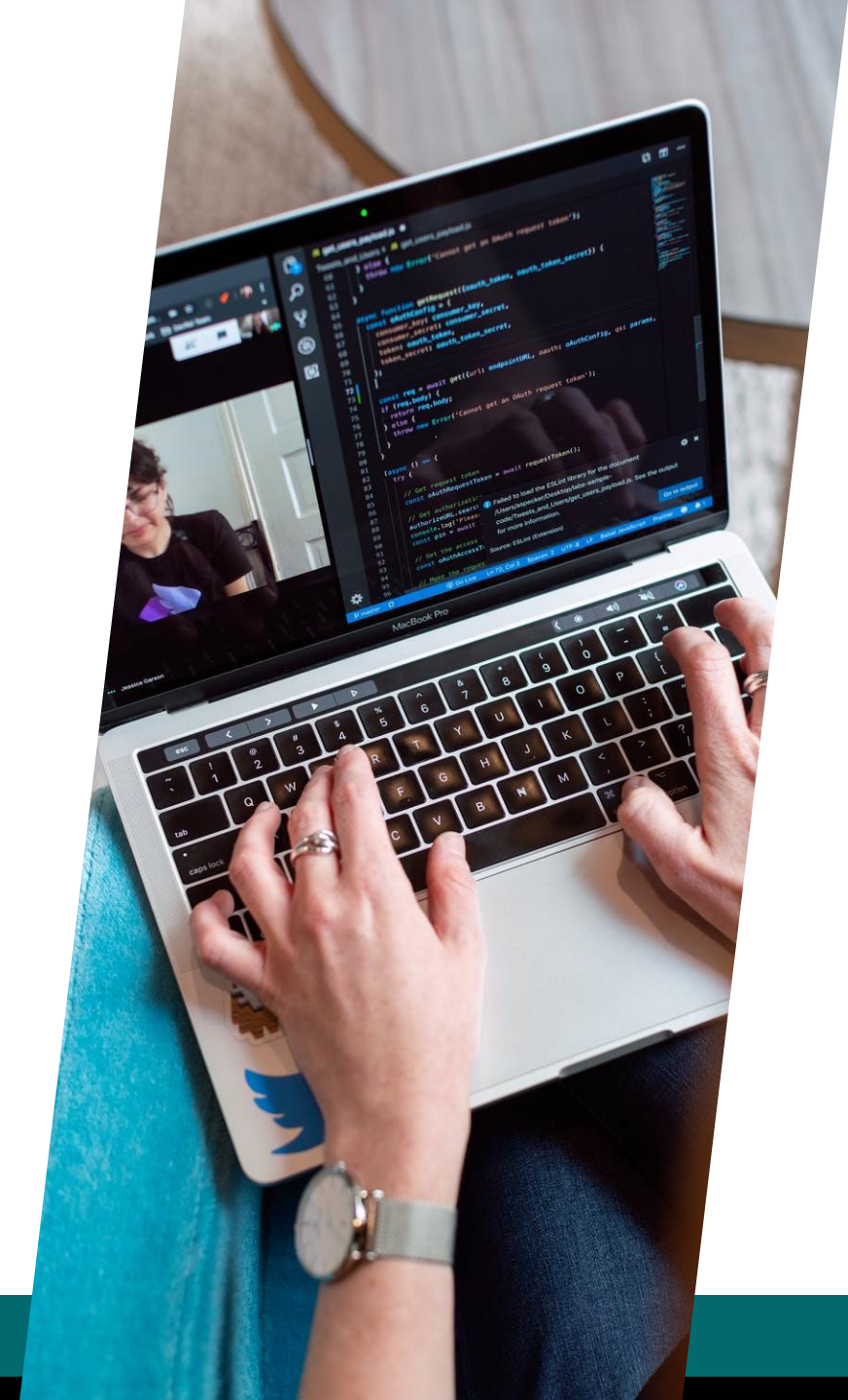


Antes de começar...

Nesse módulo você utilizou o Nodejs para criar o back-end de uma aplicação.

Os principais serviços da aplicação correspondem a implementação de um CRUD. Inserir, consultar e alterar dados já foram implementados.

A última etapa é a remoção dos dados da nossa base de dados. Vamos lá!



Criar uma rota para fazer a exclusão

1. Crie uma nova rota utilizando o método `get` para excluir um cliente.

```
175 app.get('/excluircliente', async(req,res)=>{  
176   cliente.destroy({  
177     where: {id:2}  
178   });  
179 });
```

O que acontece quando você exclui esse registro de cliente?



Excluir a partir de uma aplicação externa

1. Crie uma nova rota utilizando o método `destroy` para excluir um cliente informado como parâmetro.

```
181 app.delete('/apagarcliente/:id', (req, res)=>{
182   cliente.destroy({
183     where: {id: req.params.id}
184   }).then(function(){
185     return res.json({
186       error: false,
187       message: "Cliente excluído com sucesso!"
188     });
189   }).catch(function(erro){
190     return res.status(400).json({
191       error: true,
192       mensagem: "Erro ao tentar excluir."
193     });
194   });
195 });
```



Atualizar o Github

Até aqui tudo bem?

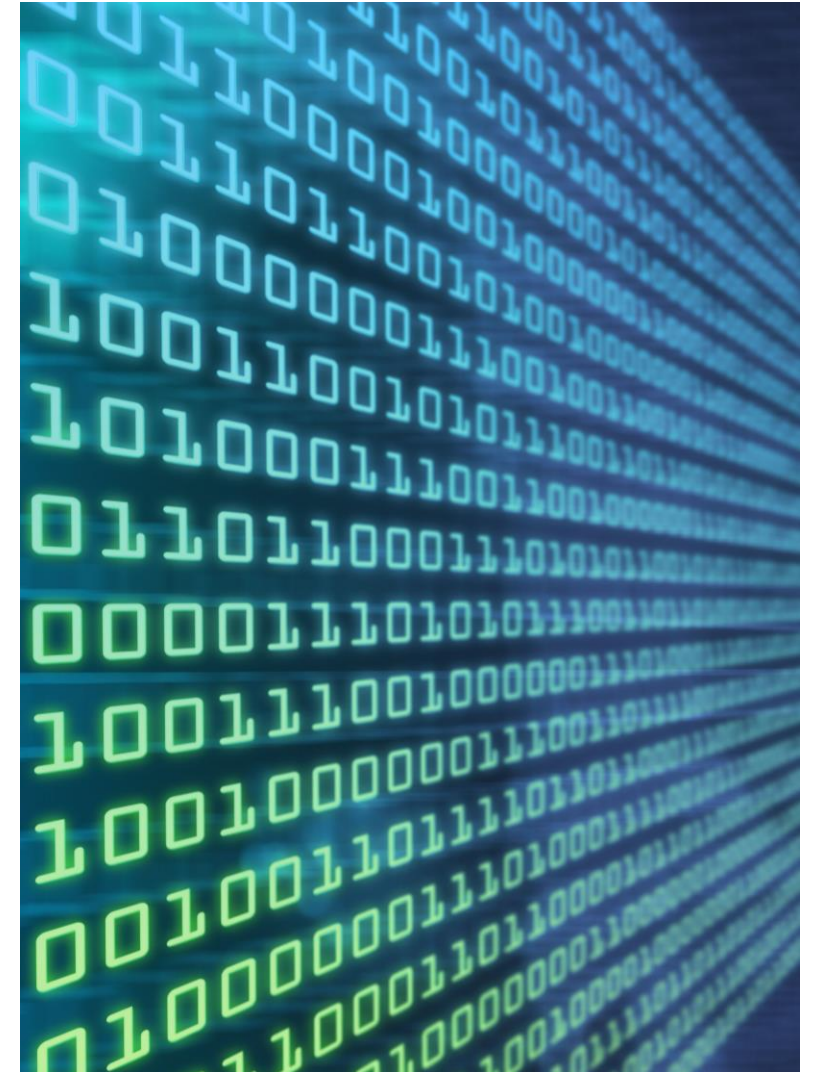
Agora é a hora de subir toda a nossa aplicação para o seu repositório.



O que vem em seguida?

No próximo módulo, além das Soft Skills, você vai utilizar o React para implementar o front-end dessa aplicação.

Vamos criar uma aplicação web que vai acessar a nossa API.





#TAKEOFF
@tiacademybrasil
www.tiacademybrasil.com.br