Dia 11 - Node.js

Sequelize - Continuação

Centro de Alta Performance - SECITECI

SECITECI
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA E INOVAÇÃO

GOVERNO DE MATO GROSSO

<Fic_Dev>
Programador de Sistemas

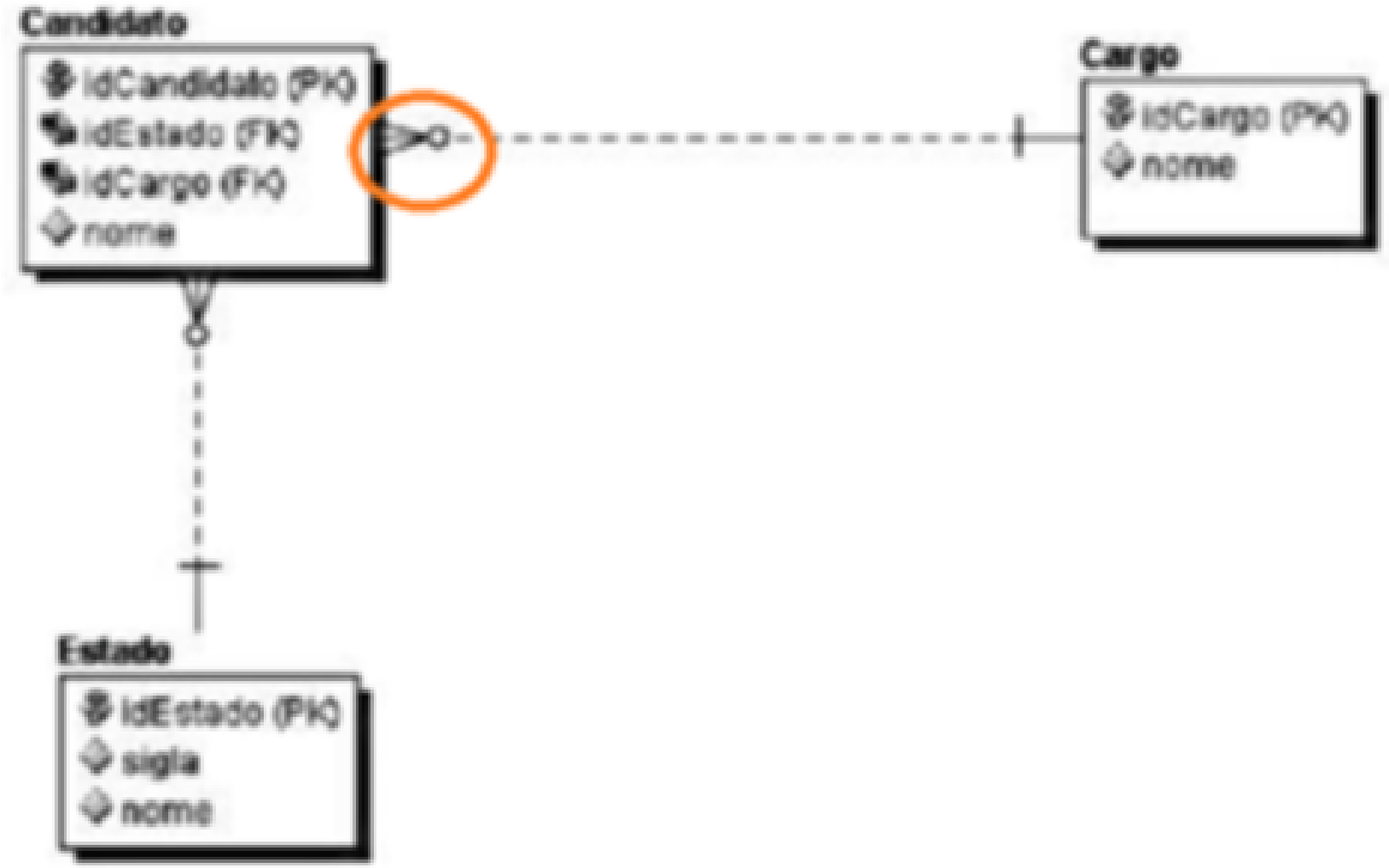
Roteiro

1. Desafio
2. Objetivo da aula
3. Conceitos
   - Models e migrations
4. Criando o projeto
5. Criando o banco
6. Criando os modelos e as migrations
7. Criando a estrutura do Banco

<Fic_Dev>
Programador de Sistemas

Desafio

Observe o modelo:



Desafio

Utilizando o sequelize, gere esse modelo no Banco de Dados.

Objetivo da disciplina

Entender o conceito de migrations.

Entender o conceito de modelos.

Saber gerar um banco de dados usando migrations e models.

Models

- **Modelos** são classes que definem a estrutura dos dados que serão manipulados pelo aplicativo.
- Os modelos do Sequelize geralmente contêm propriedades que correspondem às colunas da tabela do banco de dados

Migrations

- **Migrations** são arquivos de script que definem a estrutura da tabela do banco de dados e as alterações na estrutura da tabela ao longo do tempo.
- As Migrations podem ser usadas para criar e alterar tabelas, colunas, índices e chaves estrangeiras. Elas garantem que a estrutura do banco de dados corresponda à estrutura definida pelo modelo do aplicativo

Vamos criar o projeto

- Para iniciar o projeto crie um workspace e digite o seguinte comando no terminal :

  - `npm init -y`

- Esse comando irá criar na pasta do projeto um arquivo chamado package.json

- Vamos instalar o dotenv para configurações da aplicação:
  - `npm install dotenv`

- Na raiz do projeto, criar um arquivo .env com o seguinte conteúdo:

```
DEV_DATABASE_URL=postgres://postgres:postgres@127.0.0.1:5432/dev_db
TEST_DATABASE_URL=postgres://<user>:<password>@127.0.0.1:5432/test_db
```

- Depois instale as dependências:
  - `npm install sequelize sequelize-cli pg pg-hstore`

- Na raiz do projeto crie um arquivo com o nome de .sequelizerc e as seguinte configurações:

```
const path = require('path')

module.exports = {
'config': path.resolve('./src/database/config', 'config.js'),
'models-path': path.resolve('./src/database/models'),
'seeders-path': path.resolve('./src/database/seeders'),
'migrations-path': path.resolve('./src/database/migrations'),
}
```

*Cuidado com as aspas

- Depois de criar o arquivo, execute o seguinte comando para criar a nossa estrutura de projeto :
  - `npx sequelize-cli init`

- Seguindo o nosso arquivo de configuração do sequelize, o .sequelizerc ele iriá criar as pastas de configurações para o config, models, migrations e seeders.

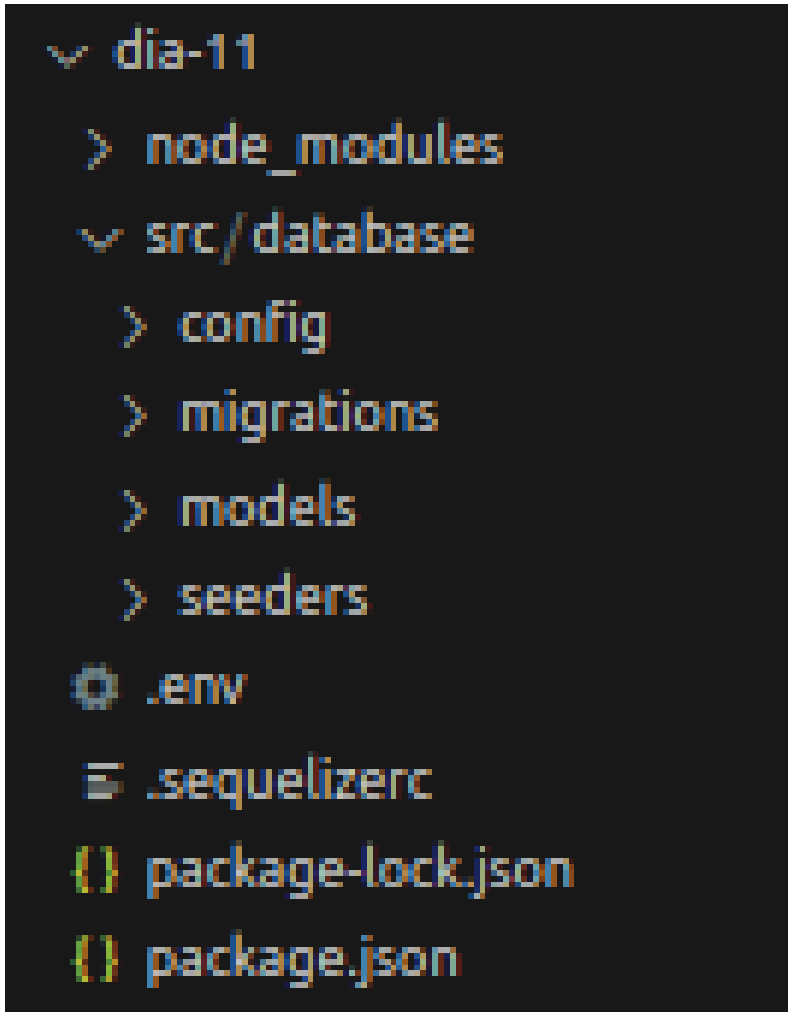
- Na pasta config terá o arquivo config.js que vai ficar a Url de conexão com o postgres. Abaixo vemos como deverá ficar a nossa configuração, o config.js:

```
require('dotenv').config();

module.exports = {
  development: {
    url: process.env.DEV_DATABASE_URL,
    dialect: 'postgres',
  },
  test: {
    url: process.env.TEST_DATABASE_URL,
    dialect: 'postgres',
  },
  production: {
    url: process.env.DATABASE_URL,
    dialect: 'postgres',
  },
};
```

- Pasta migrations e pasta models: terá a nossas migrations e models. Os models são os arquivos de criação da tabela e as migrations são alterações nessas tabelas.

Se tudo ocorreu bem até aqui, seu projeto deve ter essa estrutura:



- Após a configuração você pode executar :
  - `npx sequelize db:create`

- Esse comando irá criar um novo banco de dados chamado **dev_db**

- Agora, vamos criar todos os modelos que vamos precisar:
- Adicione npx antes de cada comando.
- npx sequelize model:generate...

```
sequelize model:generate --name Company --attributes name:STRING

sequelize model:generate --name User --attributes
email:STRING,firstName:STRING,lastName:STRING,companyId:INTEGER

sequelize model:generate --name WorkingDay --attributes
weekDay:STRING,workingDate:DATE,isWorking:BOOLEAN

sequelize model:generate --name UsersWorkingDay --attributes
userId:INTEGER,workingDayId:INTEGER
```

- Nesse exemplo temos o seguinte:
- **Company, User, WorkingDay**
- User trabalha em Company (belongsTo) 1:1
- Company tem muitos User (hasMany) 1:N
- User trabalha em muitos WorkingDay e WorkingDay tem muitos User (manyToMany) N:N
- Para criar muitos relacionamentos, precisaremos de uma tabela de junção que chamaremos de UsersWorkingDay.

- Tarefa: Faça o modelo dessa descrição

- Repare que agora você tem as migrations e os models criados.
- Ainda faltam alguns ajustes...

- Repare que agora você tem as migrations e os models criados.
- Ainda faltam alguns ajustes...
- Temos que indicar os relacionamentos nos arquivos de migração adicionando uma chave de referência em todas as chaves estrangeiras.
- Também adicionar para cada chave estrangeira um AllowNull false.
- Também temos que indicar os relacionamentos nos models.

- Nesse momento para ganharmos tempo o Professor irá fornecer os novos arquivos com as alterações mencionadas.
- DICA: Copie o conteúdo do novo arquivo e substitua o conteúdo do arquivo do seu projeto.

- É isso !
- Agora vamos rodar o comando:
  - `npx sequelize-cli db:migrate`

- Agora corra lá no Banco de Dados e dê uma olhada.