Criando a estrutura

1. Yarn init -y
2. Yarn add express
3. Yarn add nodemon -D
4. Alterar o package.jason

  "scripts": {

    "dev": "nodemon ./src/server.js"

  }

1. Criar o serve.js dentro da pasta src

const express = require('express')

const app = express();

app.listen(3001, ()=>{

*console*.log("Servidor rodando na porta 3001")

})

1. Criar uma rota

const express = require('express')

const app = express();

app.get('/', (*req*, *res*) => {

    res.send('Olá');

});

app.listen(3001, ()=>{

*console*.log("Servidor rodando na port 3001")

})

1. Banco de dados
   1. Node + mysql
   2. Node + query builder
   3. Node + sequelizer
2. Yarn add sequelize mysql2
3. Yarn add sequelize-cli -D
4. Yarn sequelize –help
5. Criar uma rquivos de rotas – routes.js

const express = require('express')

const routes = express.Router();

routes.get('/', (*req*,*res*) =>{

    return res.json({hello: "Olá"})

})

*module*.*exports* = routes;

1. Alterar o arquivo server.js

const express = require('express')

const routes = require('./routes')

const app = express();

app.use(express.json());

app.use(routes);

app.listen(3001, ()=>{

*console*.log("Servidor rodando na porta 3001")

})

1. Yarn dev
2. Configurar as credenciais de acesso a base de dados
   1. Criar uma pasta src/config
   2. Criar um arquivo database.js
      1. *module*.*exports* = {
      2. dialect: 'mysql',
      3. host: 'localhost',
      4. username: 'root',
      5. password:'123456',
      6. database: 'sqlNode',
      7. define : {
      8. timestamps: true,
      9. underscored: true
      10. }
      11. }
   3. Criar uma pasat src/database/index.js
      1. const Sequelize = require('sequelize')
      2. const dbConfig = require('../config/database')
      3. const connection = **new** *Sequelize*(dbConfig);
      4. *module*.*exports* = connection;
   4. criar o arquivo na raiz .sequelizerc
      1. const path = require('path');
      2. *module*.*exports* = {
      3. config: path.resolve(\_\_dirname,'src','config','database.js'),
      4. }
   5. yarn sequelize db:create
3. Criar a primeira tabela
   1. Alterar o arquivo .sequelizerc
      1. const path = require('path');
      2. *module*.*exports* = {
      3. config: path.resolve(\_\_dirname,'src','config','database.js'),
      4. 'migrations-path': path.resolve(\_\_dirname, 'src', 'database', 'migrations')
      5. }
   2. Criar a pasta src/database/migrations
   3. Yarn sequelize migration:create --name=create-users
   4. Criou o arquivo migrations
      1. 'use strict';
      2. *module*.*exports* = {
      3. up: (*queryInterface*, *Sequelize*) => { // dizer o que a migrations vai realizar na bd
      4. /\*
      5. Add altering commands here.
      6. Return a promise to correctly handle asynchronicity.
      7. Example:
      8. return queryInterface.createTable('users', { id: Sequelize.INTEGER });
      9. \*/
      10. },
      11. down: (*queryInterface*, *Sequelize*) => { // se der alguma coisa errada o que tenho que fazer
      12. /\*
      13. Add reverting commands here.
      14. Return a promise to correctly handle asynchronicity.
      15. Example:
      16. return queryInterface.dropTable('users');
      17. \*/
      18. }
      19. };
   5. Alterar o migrations
      1. 'use strict';
      2. *module*.*exports* = {
      3. up: (*queryInterface*, *Sequelize*) => { // dizer o que a migrations vai realizar na bd
      4. return queryInterface.createTable('users', {
      5. id: {
      6. type: Sequelize.INTEGER,
      7. primaryKey: true,
      8. autoIncrement: true,
      9. allowNull: false,
      10. },
      11. name: {
      12. type: Sequelize.STRING,
      13. allowNull: false,
      14. },
      15. email: {
      16. type: Sequelize.STRING,
      17. allowNull: false,
      18. },
      19. created\_at:{
      20. type: Sequelize.DATE,
      21. allowNull: false,
      22. },
      23. updated\_at:{
      24. type: Sequelize.DATE,
      25. allowNull: false,
      26. }
      27. });
      29. },
      30. down: (*queryInterface*, *Sequelize*) => { // se der alguma coisa errada o que tenho que fazer
      31. return queryInterface.dropTable('users');
      33. }
      34. };
   6. Yarn sequelize db:migrate
   7. Para voltar yarn sequelize db:migrate:undo
4. Criar a pasta model dentro do src
   1. Criar o users.js
      1. const {Model, DataTypes} = require('sequelize');
      2. class User extends *Model*{
      3. static init(*sequelize*){
      4. *super*.init({
      5. name: DataTypes.STRING,
      6. email: DataTypes.STRING,
      7. },  {
      8. sequelize
      9. })
      10. }
      11. }
      12. *module*.*exports* = User;
   2. Arquivo index.js da pasta database
      1. const Sequelize = require('sequelize')
      2. const dbConfig = require('../config/database')
      3. const User = require('../models/users')
      4. const connection = **new** *Sequelize*(dbConfig);
      5. User.init(connection);
      6. *module*.*exports* = connection;
5. Alterar o server.js
   * 1. const express = require('express')
     2. const routes = require('./routes')
     3. require('./database')
     4. const app = express();
     5. app.use(express.json());
     6. app.use(routes);
     7. app.listen(3001, ()=>{
     8. *console*.log("Servidor rodando na porta 3001")
     9. })
6. Criar uma pasta src/controllers
   * 1. const User = require('../models/users')
     2. *module*.*exports* = {
     3. async store (*req*,*res*) {
     4. const {name, email} = req.body;
     6. const user = await User.create({name, email});
     7. return res.json(user)
     8. }
     10. };
   1. Alterar o arquivo de rotas
      1. const express = require('express')
      2. const UserController = require('./controllers/UserController')
      3. const routes = express.Router();
      4. routes.get('/', (*req*,*res*) =>{
      5. return res.json({hello: "Olá"})
      6. })
      7. routes.post('/users', UserController.store);
      8. *module*.*exports* = routes;
7. Criar o método de listar todos no BD
   1. Alterar o userController criando o método indexAll
      1. const User = require('../models/User')
      2. *module*.*exports* = {
      3. async store (*req*,*res*) {
      4. const {name, email} = req.body;
      5. *console*.log(name, email)
      6. const user = await User.create({name, email});
      7. return res.json(user)
      8. },
      9. async indexAll(*req*,*res*){
      10. const users = await User.findAll();
      11. return res.json(users);
      12. }
      14. };
   2. Adicionar no arquivo Rotas
      1. routes.get('/users', UserController.indexAll);
   3. Testar o get no insomnia
8. Relacionamentos
   1. Usuario pode ter vários endereços no banco de dados
   2. yarn sequelize migration:create --name=create-addresses
   3. criar a migration endereço
      1. 'use strict';
      2. *module*.*exports* = {
      3. up: (*queryInterface*, *Sequelize*) => {
      4. return queryInterface.createTable('addresses', {
      5. id: {
      6. type: Sequelize.INTEGER,
      7. primaryKey: true,
      8. autoIncrement: true,
      9. allowNull: false,
      10. },
      11. user\_id:{
      12. type: Sequelize.INTEGER,
      13. allowNull: false,
      14. references: {model: 'users', key: 'id'},
      15. onUpdate: 'CASCADE',
      16. onDelete: 'CASCADE',
      17. },
      18. CEP: {
      19. type: Sequelize.STRING,
      20. allowNull: false,
      21. },
      22. rua: {
      23. type: Sequelize.STRING,
      24. allowNull: false,
      25. },
      26. numero: {
      27. type: Sequelize.INTEGER,
      28. allowNull: false,
      29. },
      30. created\_at:{
      31. type: Sequelize.DATE,
      32. allowNull: false,
      33. },
      34. updated\_at:{
      35. type: Sequelize.DATE,
      36. allowNull: false,
      37. }
      38. });
      40. },
      41. down: (*queryInterface*, *Sequelize*) => {
      42. return queryInterface.dropTable('addresses');
      43. }
      44. };
   4. Yarn sequelize db:migrate
   5. Criar o model
      1. const {Model, DataTypes} = require('sequelize');
      2. class Adsress extends *Model*{
      3. static init(*sequelize*){
      4. *super*.init({
      5. CEP: DataTypes.STRING,
      6. rua: DataTypes.STRING,
      7. numero: DataTypes.INTEGER,
      8. },  {
      9. sequelize
      10. })
      11. }

static associate(*models*){

*this*.belongsTo(models.User, {foreignKey : 'user\_id', as : 'possuiUsuario'}) //endereço possui um dono

    }

* + 1. }
    2. *module*.*exports* = Address;
  1. Importar o model dentro do arquivo index do banco de dados
     1. Migrations/index
     2. const Sequelize = require('sequelize')
     3. const dbConfig = require('../config/database')
     4. const User = require('../models/User')
     5. const Address = require('../models/Address')
     6. const connection = **new** *Sequelize*(dbConfig);
     7. User.init(connection);
     8. Address.init(connection)
     9. Address.associate(connection.models)
     10. *module*.*exports* = connection;
  2. Alterar a rota de adicionar endereço
     1. const AddressController = require('./controllers/AddressController')
     2. routes.post('/users/:user\_id/address', AddressController.store);
  3. Criar o AddressController
     1. const Address = require('../models/Address')
     2. const User = require('../models/User')
     3. *module*.*exports* = {
     4. async store(*req*, *res*) {
     5. const { CEP, rua, numero } = req.body;
     6. *console*.log(CEP, rua, numero)
     7. const { user\_id } = req.params;
     8. //verificar se o usuario existe
     9. const user = await User.findByPk(user\_id)
     10. if (!user) {
     11. return res.status(400).json({ error: 'Usuario não encontrado' })
     12. }
     13. const address = await Address.create({
     14. CEP,
     15. rua,
     16. numero,
     17. user\_id
     18. });
     19. return res.json(address)
     20. },
     21. };

1. Buscar endereço de acrodo com o usuário
   1. Criar uma rota do tipo get
      1. routes.get('/users/:user\_id/addresses', AddressController.index);
   2. Realizar a associação de um usuário possu vários endereços
      1. const {Model, DataTypes} = require('sequelize');
      2. class User extends *Model*{
      3. static init(*sequelize*){
      4. *super*.init({
      5. name: DataTypes.STRING,
      6. email: DataTypes.STRING,
      7. },  {
      8. sequelize
      9. })
      10. }
      11. static associate(*models*){
      12. *this*.hasMany(models.Address, {foreignKey : 'user\_id', as : 'enderecos'})
      13. }
      14. }
      15. *module*.*exports* = User;
   3. Acrescentar o método associate no index.js
      1. User.associate(connection.models)
   4. Alterar o AddressControllers
      1. const Address = require('../models/Address')
      2. const User = require('../models/User')
      3. *module*.*exports* = {
      4. async index(*res*,*req*){
      5. const { user\_id} = req.params;
      6. const user = await User.findByPk(user\_id,{
      7. include: { association: 'enderecos'}
      8. })

      11. //const addresses = await Address.findAll({where: {user\_id} })
      12. //return res.json(addresses)
      13. return res.json(user)
      14. },
      15. async store(*req*, *res*) {
      16. const { cep, rua, numero } = req.body;
      17. *console*.log(cep, rua, numero)
      18. const { user\_id } = req.params;
      19. //verificar se o usuario existe
      20. const user = await User.findByPk(user\_id)
      21. if (!user) {
      22. return res.status(400).json({ error: 'Usuario não encontrado' })
      23. }
      24. const address = await Address.create({
      25. cep,
      26. rua,
      27. numero,
      28. user\_id
      29. });
      30. return res.json(address)
      31. },
      32. };
      33. Alterar
2. Criar tabela N-N
   1. yarn sequelie migration:create --name=create-techs
   2. editar o migration techs
      1. 'use strict';
      2. *module*.*exports* = {
      3. up: (*queryInterface*, *Sequelize*) => {
      4. return queryInterface.createTable('techs', {
      5. id: {
      6. type: Sequelize.INTEGER,
      7. primaryKey: true,
      8. autoIncrement: true,
      9. allowNull: false,
      10. },
      11. name:{
      12. type: Sequelize.STRING,
      13. allowNull: false,
      14. },
      15. created\_at:{
      16. type: Sequelize.DATE,
      17. allowNull: false,
      18. },
      19. updated\_at:{
      20. type: Sequelize.DATE,
      21. allowNull: false,
      22. },
      23. });
      24. },
      25. down: (*queryInterface*, *Sequelize*) => {
      27. return queryInterface.dropTable('users');
      29. }
      30. };
   3. yarn sequelie migration:create --name=create-user\_techs
   4. Alterar o migration user\_techs
      1. 'use strict';
      2. *module*.*exports* = {
      3. up: (*queryInterface*, *Sequelize*) => {
      4. return queryInterface.createTable('user\_techs', {
      5. id: {
      6. type: Sequelize.INTEGER,
      7. primaryKey: true,
      8. autoIncrement: true,
      9. allowNull: false,
      10. },
      11. user\_id:{
      12. type: Sequelize.INTEGER,
      13. allowNull: false,
      14. references: {model: 'users', key: 'id'},
      15. onUpdate: 'CASCADE',
      16. onDelete: 'CASCADE',
      17. },
      18. tech\_id:{
      19. type: Sequelize.INTEGER,
      20. allowNull: false,
      21. references: {model: 'techs', key: 'id'},
      22. onUpdate: 'CASCADE',
      23. onDelete: 'CASCADE',
      24. },
      25. created\_at: {
      26. type: Sequelize.DATE,
      27. allowNull: false,
      28. },
      29. updated\_at: {
      30. type: Sequelize.DATE,
      31. allowNull: false,
      32. },
      33. });
      34. },
      35. down: (*queryInterface*, *Sequelize*) => {
      36. return queryInterface.dropTable('user\_techs');
      38. }
      39. };
   5. yarn sequelize db:migrate
   6. Criar um model techs
      1. const {Model, DataTypes} = require('sequelize');
      2. class Tech extends *Model*{
      3. static init(*sequelize*){
      4. *super*.init({
      5. name: DataTypes.STRING,
      6. },  {
      7. sequelize,
      8. tableName: 'techs'
      9. })
      10. }
      11. static associate(*models*){
      12. *this*.belongsToMany(models.User, {
      13. foreignKey : 'tech\_id',
      14. through: 'user\_techs',
      15. as : 'users'})
      16. }
      17. }
      18. *module*.*exports* = Tech;
   7. Alterar o models Usuario
      1. const {Model, DataTypes} = require('sequelize');
      2. class User extends *Model*{
      3. static init(*sequelize*){
      4. *super*.init({
      5. name: DataTypes.STRING,
      6. email: DataTypes.STRING,
      7. },  {
      8. sequelize
      9. })
      10. }
      11. static associate(*models*){
      12. *this*.hasMany(models.Address, {foreignKey : 'user\_id', as : 'enderecos'})
      13. *this*.belongsToMany(models.Tech, {
      14. foreignKey : 'user\_id',
      15. through: 'user\_techs',
      16. as : 'techs'})
      17. }
      18. }
      19. *module*.*exports* = User;
   8. Inicializar o model de tech no arquivo index.js
      1. const Sequelize = require('sequelize')
      2. const dbConfig = require('../config/database')
      3. const User = require('../models/User')
      4. const Address = require('../models/Address')
      5. const Tech = require('../models/Tech')
      6. const connection = **new** *Sequelize*(dbConfig);
      7. User.init(connection);
      8. Address.init(connection)
      9. Tech.init(connection)
      10. Address.associate(connection.models)
      11. User.associate(connection.models)
      12. Tech.associate(connection.models)
      13. *module*.*exports* = connection;
   9. Criar as rotas de get e store
      1. const TechController = require('./controllers/TechController')
      2. routes.post('/users/:user\_id/techs', TechController.store);
      3. routes.get('/users/:user\_id/techs', TechController.index);
   10. controlerTech – store
       1. const Tech = require('../models/Tech')
       2. const User = require('../models/User')
       3. *module*.*exports* = {
       4. async index(*req*,*res*){
       5. },
       6. async store(*req*, *res*) {
       7. const {user\_id} = req.params;
       8. const {name} = req.body;
       9. const user = await User.findByPk(user\_id)
       10. if (!user) {
       11. return res.status(400).json({ error: 'Usuario não encontrado' })
       12. }
       13. //created bolean que define se a tecnologia foi criada ou não
       14. const [tech, created] = await Tech.findOrCreate({
       15. where : {name}
       16. })
       17. await user.addTech(tech);
       18. return res.json(tech)
       19. },
       20. };
   11. Deletar uma tecnologia do usuário
       1. async delete(*req*,*res*){
       2. const {user\_id} = req.params;
       3. const {name} = req.body;
       4. const user = await User.findByPk(user\_id)
       5. if (!user) {
       6. return res.status(400).json({ error: 'Usuario não encontrado' })
       7. }
       8. const tech = await Tech.findOne({
       9. where : {name}
       10. })
       11. await user.removeTech(tech)
       12. return res.json({mensage : "Removida com sucesso"})
       13. }
   12. Adicionar a rota delete
       1. routes.delete('/users/:user\_id/techs', TechController.delete);
   13. Listagem