ROTEIRO PARA CRIAR AUTENTICAÇAO DE UMA API REST COM JWT E BANCO DE DADOS

1. NA pasta APIDEGAMES2
2. Realize a instalação das bibliotecas express, mysql2, sequelize, cors e jsowebtoken
3. Crie a pasta database
4. Na pasta database
5. Crie o arquivo database.js
6. No arquivo database.js importe o sequelize crie uma constante para realizar a conexão com o banco de dados.
7. No fim exporte a conexão
8. const Sequelize = require("sequelize");
9. const connection = **new** Sequelize('testesbd','root','123456',{
10. host:'localhost',dialect:'mysql'
11. });
12. *module*.*exports* = connection;
13. Crie a pasta Model que terá os models que serão as tabelas do banco de dados
14. Crie o arquivo Game.js
15. Abra o arquivo Game.js, importe o sequelize e connection.
16. Crie a constante Game e define as propriedades da tabela games
17. E exporte a constante Game
18. const Sequelize = require("sequelize");
19. const connection = require("../database/database");
20. //const User = require("./User");
21. const Game = connection.define('games',{
22. title: {
23. type:Sequelize.STRING,
24. allowNull: false},
25. year:{
26. type:Sequelize.INTEGER,
27. allowNull: false},
28. price:{
29. type:Sequelize.DOUBLE,
30. allowNull: false}
31. });
32. //Game.sync({force:true});
33. *module*.*exports* = Game;
34. Crie o arquivo User.js
35. Abra o arquivo User.js e importe o sequelize e a connection
36. Crie a constante User e defina as propriedades da tabela Users
37. Exporte a constante User
38. Na pasta APIDEGAMES2
39. Crie o arquivo index.js
40. No arquivo index.js
41. Importe as bibliotecas que serão utilizadas
42. //importando as bibliotecas
43. const express = require("express");
44. const cors = require("cors");
45. const sequelize = require("sequelize");
46. importe a conexão e realize a autenticação com o banco de dados
47. //importando a connection
48. const connection = require('./database/database');
49. //realizando autenticação no banco de dados
50. connection.authenticate().then(()=>{
51. console.log("Conexão OK");
52. }).catch(*err*=>{
53. console.log('Erro: '+*err*);
54. });
55. Importe os models que serão utilizados
56. //importando os models
57. const User = require('./model/User');
58. const Game = require('./model/Game');
59. Crie a instancia do express
60. //criando o app
61. const app = express();
62. importe a biblioteca jsonwebtoken e crie uma chave secreta que será utilizada na autenticação
63. //instanciando a biblioteca jsonwebtoken para autenticação
64. const jwt = require("jsonwebtoken");
65. //criando uma chave secreta para o jwt para realizar a autenticação
66. const secretKeyJwt = "ffhhae@#$¨%fdd!@#4";
67. Realize a configuração básica do express
68. /inicando o express com configuração básica
69. app.use(express.urlencoded({extended:false}));
70. app.use(express.json());
71. Crie função de autenticação que será o middleware das rotas
72. function auth(*req*, *res*, *next*){
73. //pegando o header de autorização da requisição
74. const authToken = *req*.headers['authorization']
75. //validando o token
76. if(authToken != undefined){
77. //cortando o token
78. const bearer = authToken.split(' ');
79. //pegando somente o token
80. var token = bearer[1];
81. //verificando se o token é valido
82. jwt.verify(token,secretKeyJwt,(*err*, *data*)=>{
83. if(*err*){
84. *res*.status(401);
85. *res*.json({err:"Token inválido!"});
86. }else{
87. //passando variáveis para a requisição
88. //, estes dados poderão ser acessados
89. //nas rotas que forem autenticadas
90. *req*.token = token;
91. *req*.logedUser = {id: *data*.id, email: *data*.email};
92. //console.log(data);
93. //se autenticação foi realizada com sucesso a função next que irá
94. //realizar a requisição do usuário
95. next();
96. }
97. });
98. // console.log(bearer);
99. }else{
100. *res*.status(401);
101. *res*.json({err:"Token inválido"});
102. }
103. //console.log(authToken);

106. }
107. Criando a rota para carregar os games na página
108. /\*\*primeira rota no primeiro end point que deverá listar todos
109. \* os games do sistema, incluindo o midleware de autenticação auth
110. \*/
111. app.get('/games',auth,(*req*,*res*)=>{
112. var {name,email,password} = *req*.body;
113. //consulta os games no banco de dados
114. Game.findAll().then((*games*)=>{
115. if(*games* !=undefined){
116. *res*.status(200);
117. // res.sendStatus(200);
118. //json que irá mostrar os games do banco de dados
119. //e o usuário que foi autenticado na funcao auth
120. *res*.json({user: *req*.logedUser, games: *games*});
121. console.log("Jogos carregados com sucesso.");
122. }
123. }).catch(*err*=>{
124. *res*.status(404);
125. // res.sendStatus(404);
126. *res*.json({erro: "Erro :", err:*err*});
127. console.log('Erro: '+ *err*);
128. });
129. // res.statusCode = 200;
130. });
131. Criar a rota para localizar um game pelo ID
132. \*\*rota para devolver ao usuário o game pelo ID \*/
133. app.get('/game/:id',auth,(*req*,*res*)=>{
135. //tratando o id verificando se é numero
136. if(isNaN(*req*.params.id)){
137. *res*.sendStatus(400);
138. }else{
139. var id = parseInt(*req*.params.id);
141. //var game = DB.games.find( game =>  game.id == id );
142. Game.findOne({
143. where:{id:id}
144. }).then((*game*)=>{
145. if(*game* != undefined){
146. *res*.status(200);
147. *res*.json(*game*);
148. }else{
149. *res*.sendStatus(404);
150. }
151. }).catch(*err*=>{
152. console.log("Erro: "+*err*);
153. });
154. }
155. });
156. Criar rota para cadastrar um novo game
157. /\*\*novo endpoint para cadastrar um novo game \*/
158. app.post('/game',auth,(*req*,*res*)=>{
160. //utilizando a destruct para facilitar a manipulação de dados
161. var {title, year, price} = *req*.body;
162. //o ideal é validar todos os dados conforme as regras no negocio
163. //e retornar o status code 400 caso não se enquadre nas regras
165. //adicinando dados no banco de dados
166. Game.create({
167. title: title,
168. year: year,
169. price: price
170. }).then(()=>{
171. *res*.status(200);
172. console.log("Jogo criado com sucesso.");
173. *res*.sendStatus(200);
174. }).catch(*err*=>{
175. *res*.sendStatus(401);
176. console.log("Erro: "+*err*);
177. });
178. });
179. Criar a rota para apagar um game pelo ID
180. /\*\*novo endpoint para deletar um jogo \*/
181. app.delete('/game/:id',auth,(*req*,*res*)=>{
182. /\*\*validação \*/
183. if(isNaN(*req*.params.id)){
184. *res*.sendStatus(400);
185. }else{
186. var id = parseInt(*req*.params.id);
187. var game = Game.findOne({where:{id:id}});
188. if(game != undefined){
189. Game.destroy({
190. where:{id:id}
191. }).then(()=>{
192. *res*.status(200);
193. console.log("Game apagado");
194. *res*.json({g:"Game apagado com sucesso!"});
195. }).catch();
196. }else{
197. *res*.sendStatus(404);
198. }
199. }
200. });
201. Criar rota para alterar o game pelo ID
202. \*\*endpois para edição de registros será criado uma rota
203. \* utilizando o PUT
204. \*/
205. app.put('/game/:id',auth,(*req*,*res*)=>{
206. /\*\*validação \*/
207. if(isNaN(*req*.params.id)){
208. *res*.sendStatus(400);
209. }else{
210. var id = parseInt(*req*.params.id);
211. /\*\*localiza o jogo pelo id
212. \*/
213. //var game = DB.games.find( g=> g.id == id);
214. var game = Game.findOne({where:{id:id}})
216. if(game != undefined){
217. //peqando os dados que estão na requisição
218. var {title, price, year} = *req*.body;
220. if(title == undefined){
221. title = game.title;
222. }
223. if(year == undefined){
224. year = game.year;
225. }
226. if(price = undefined){
227. price = game.price;
228. }
229. Game.update({
230. title: title,
231. year: year,
232. price: price
233. },{where:{id:id}}).then(()=>{
234. *res*.status(200);
235. *res*.sendStatus(200);
236. }).catch(*err*=>{
237. *res*.status(401);
238. *res*.sendStatus(401);
239. });
240. }
241. }
242. });
243. Criar rota para a autenticação do usuário e geração do token.
244. /\*\*rota para autenticação \*/
245. app.post("/auth",(*req*,*res*)=>{
246. var {id, email, password} = *req*.body;
247. //validações
248. if(email != undefined){
249. User.findOne({where:{email:email}}).then((*user*)=>{
250. if(*user* != undefined){
251. if(*user*.password === password){
252. /\*\*deve-se passar as informações necessárias
253. \* , a chave secreta e o prazo que irá expirar
254. \* o token
255. \*/
256. jwt.sign(
257. {
258. id: *user*.id,
259. email: *user*.email
260. },
261. secretKeyJwt,
262. {expiresIn:'48h'},(*err*, *token*)=>{
263. if(*err*){
264. *res*.status(400);
265. *res*.json({err: "Falha interna"});
266. }else{
267. *res*.status(200);
268. *res*.json({token: *token*});
269. }
270. })
272. }else{
273. *res*.status(401);
274. *res*.json({err: "Email ou senha inválidos1", p: *user*.password , p2: password});
275. }
277. }else{
278. *res*.status(404);
279. *res*.json({err: "Email ou senha inválidos2"});
280. }
281. }).catch(*err*=>{
282. *res*.status(401);
283. *res*.json({err:"Erro :"+*err*});
284. });
285. }else{
286. *res*.status(400);
287. *res*.json({err: "O email enviado é inválido."});
288. }
289. });
290. Configurar o listen do aplicativo
291. app.listen(4040,()=>{
292. console.log("API rodando");
293. });