**راهنمای استفاده از SQLite در Nodejs با Promise و الگوی Repository**

توی این راهنما با هم می‌خواهیم یک پروژه‌ی نمونه آماده کنیم که برامون دسترسی متمرکزی به SQLite توی محیط Node.js با الگوی Repository فراهم بیاره. از Promiseها هم استفاده میکنیم تا در طول کار با برنامه بتونیم از Promise chain ها استفاده کنیم.

دیتابیس SQLite یک دیتابیس سبک، Cross platform و محبوبه. هنوز هم به شدت کارا و توی کاربردهای زیادی از جمله برنامه‌های موبایلی از این دیتابیس استفاده میشه. جایگاه SQLite به نسبت مابقی دیتابیس‌ها رو می‌تونید [اینجا](https://db-engines.com/en/ranking)مشاهده کنید.

الگوی Repository هم امکان دسترسی متمرکز و یکپارچه به داده‌ها رو برامون فراهم میکنه. با استفاده از این الگو تنها یک نقطه‌ی اتصال به دیتابیس در طول برنامه داریم و لایه دسترسی به دیتا رو می‌تونیم از بقیه برنامه جدا کنیم. برای مطالعه بیشتر می‌تونید [اینجا](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design)رو مشاهده کنید.

امروز قرار نیست که به تئوری‌ها بپردازیم. پس بریم به ادامه کار به صورت عملی بپردازیم.

**ایجاد پروژه**

توی سیستم یک پوشه با نام node-sqlite-tutorial (یا هر نام دلخواه خودتون) ایجاد کنید. برای نصب کتابخانه‌های مورد نیاز و تشکیل ساختار پروژه، با git bash (و یا cmd) ابتدا به محل ایجاد پروژه برید و در ادامه دستور زیر رو وارد کنید.

npm init

و بعد توی جواب سوال مقادیر زیر رو وارد کنید (به دلخواه خودتون میتونید بعضی از مقادیر رو تغییر بدید)

name: (app) sqlite-tutorial

version: (0.0.0) 1.0.0

description: A Simple Project To Use Sqlite in Node

entry point: (index.js) app.js

test command:

git repository:

keywords: sqlite, node

author: Your Name

license: MIT

در نهایت فایل package.json ما باید چیزی شبیه به این بشه.

{

"name": "sqlite-tutorial",

"version": "1.0.0",

"description": "A Simple Project To Use Sqlite in Node",

"main": "app.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [ "sqlite", "node" ],

"author": "Hosein Mansouri",

"license": "MIT"

}

تنها کتابخونه‌ای که امروز نیاز داریم کتابخونه‌ی [sqlite3](https://www.npmjs.com/package/sqlite3)است. با دستور زیر این کتابخونه رو به پروژه اضافه کنید.

npm install --save sqlite3

**جداول پروژه**

توی این پروژه ما دو تا جدول پروژه و وظیفه داریم. هر پروژه‌ای شامل چندین و چند وظیفه مختلف میشه. ساختار جداول پروژه به شرح زیر است:

Project: { // جدول پروژه

Id int identity, // کلید اصلی جدول

Name nvarchar(250) // نام پروژه

}

Task: { // جدول وظیفه

Id int identity, //کلید اصلی جدول

ProjectId int, // کلید خارجی جدول پروژه

Name nvarchar(250), //نام وظیفه

Description nvarchar(1000), //توضیحات

IsCompleted bit // مشخص نمودن اتمام کار

}

حالا که جدول‌ها رو می‌شناسیم با هم سراغ پیاده‌سازی اونها میریم.

**پیاده‌سازی کلاس Context**

یک پوشه به نام context به همراه یک فایل به نام context.js ایجاد کنید. محتوای این فایل دسترسی متمرکز به دیتابیس رو برای ما فراهم میکنه.

const sqlite3 = require("sqlite3"); //ایمپورت کردن کتابخانه کار با اس کیو ال لایت

class Context {

constructor(dbFilePath) { //سازنده ی کلاس

this.db = new sqlite3.Database(dbFilePath, err => { //ایجاد شی دی بی

if (!err) console.log("Connected to Database");

else  console.log("Could not connected to database", err);

}

});

اینجا یک کلاس با نام Context ساختیم. داخل این کلاس تمام عملیات مورد نیاز دیتابیس رو انجام میدیم. توی تابع سازنده این کلاس آدرس دیتابیس رو می‌گیریم و برای اتصال به دیتابیس تلاش می‌کنیم. در صورت موفقیت یا عدم موفقیت هم پیغام مناسب رو توی کنسول نشون میدیم. همچنین یک شی با نام db هم برای ارتباط با دیتابیس توی این متد ایجاد می‌کنیم.

کتابخونه‌ی SQLite3 تابع‌های مختلفی داره. سه از مهمترین تابع‌های این ها هستند:

* تابع run: این تابع وظیفه اجرای هر تغییری بر روی دیتابیس رو بر عهده داره. این تغییر میتونه ایجاد یک جدول باشه یا اینکه دیتاهای یک جدول رو با دستورات insert و update یا delete تغییر بده.
* تابع get: این تابع یک سطر از نتیجه ی کوئری رو برای من برمیگردونه. (برای گرفتن اطلاعات بر اساس id استفاده میشه)
* تابع all: این تابع چندین سطر از جواب رو برای ما میتونه برگردونه.

این الگو رو توی کتابخونه‌های دیگه مثل GraphQL هم داریم که دستوراتی که دیتابیس رو تغییر میدن با دستورات فقط خواندنی هستند جدا شدند.

**پیاده‌سازی دستورات کار با دیتابیس**

الگوی متد run به شکل زیر هست.

db.run('SOME SQL QUERY', [param1, param2], (err) => {

if (err) console.log('ERROR!', err)

})

ورودی اول این متد، کوئری مورد نظر ماست. بخش دوم به صورت اختیاری میتونه شاید یک یا چند تا پارامتر باشه. دلیل استفاده از پارامترها هم جلوگیری از sql injection توی برنامه است.

متد run رو به شکل زیر پیاده سازی میکنیم. متد اصلی نتیجه کار رو به صورت callback بر میگردونه. به خاطر اینکه از جهنم callback فرار کنیم ما این متد را با استفاده از کلمه کلیدی Promise و الگوی خط دوم کد به یک تابعی Promise شده تغییر میدیم. این کدها رو بعد از تابع contractor به فایلمون اضافه میکنیم.

this.run = (sql, params = []) => {

return new Promise((resolve, reject) => {

this.db.run(sql, params, function(err) {

if (err) reject(err);  //برگردوندن خطا

else resolve({ id: this.lastID });

});

});};

الگوی تابع‌های get و all هم شبیه تابع run هست. به همین دلیل این دو تابع از پیاده‌سازی مشابه‌ای استفاده میکنند. بعد از متد run خطوط زیر رو میتونید به فایل context اضافه کنید.

this.get = (sql, params = []) => {

return new Promise((resolve, reject) => {

this.db.get(sql, params, (err, result) => {

if (err) reject(err);

else resolve(result);

});

});};

this.all = (sql, params = []) => {

return new Promise((resolve, reject) => {

this.db.all(sql, params, (err, result) => {

if (err) reject(err);

else resolve(result);

});

});};

متدهای اصلی کار با دیتابیس در طول برنامه همین سه متد هستند. اینجا کار با فایل context تموم میشه و میتونیم سراغ نوشتن فایل‌های repository بریم.

**پیاده‌سازی لایه Repository**

این لایه وظیفه جداسازی لایه‌ سرویس رو از لایه ارتباط با دیتا داره. اینجوری لایه سرویس هیچ وقت به طور مستقیم با دیتابیس در ارتباط نیست و در صورت تغییر دیتابیس تنها با تغییر لایه repository همه چیز مثل قبل کار خواهد کرد.

در root پروژه یک پروژه با نام repository ایجاد کنید و در ادامه داخل این پوشه دو فایل زیر رو بسازید.

* project-repository.js
* task-repository.js

در ابتدا سراغ فایل project-repository میریم.

class ProjectRepository {

constructor(context) { //تابع سازنده

this.context = context;

}

module.exports = ProjectRepository;

توی سازنده این کلاس ما شی context رو به صورت ورودی میگیریم. و در تمام تابع‌های بعدی این کلاس از این شی برای کار با دیتابیس استفاده میکنیم. اینجوری راه رو برای تغییرات آینده‌ی برنامه باز نگه می‌داریم.

در ادامه هم کدهای زیر را بعد از تابع سازنده به این کلاس اضافه کنید.

createTable() { //ساخت جدول

const sqlQuery = `CREATE TABLE IF NOT EXISTS Project

(Id INTEGER PRIMARY KEY AUTOINCREMENT,Name TEXT)`;

return this.context.run(sqlQuery);

}

insert(name) { //درج دیتا

return this.context.run(`INSERT INTO Project (Name) VALUES (?)`, [name]);

}

update(project) {//بروزرسانی دیتا

const { id, name } = project;

return this.context.run(`UPDATE Project SET Name = ? WHERE Id = ?`, [name,id]);

}

delete(id) {//حذف

return this.context.run(`DELETE FROM Project WHERE Id = ?`, [id]);

}

getById(id) {// گرفتن با ای دی

return this.context.get(`SELECT \* FROM Project WHERE Id = ?`, [id]);

}

getAll() { // گرفتن کل اطلاعات

return this.context.all(`SELECT \* FROM Project`);

}

تابع‌های این کلاس شامل درج، بروزرسانی، حذف، گرفتن اطلاعات بر اساس Id و گرفتن تمام رکوردهای جدول Project هست. توی تمامی این تابع‌ها شی context صدا زده شده و اجرای کوئری‌ها به این کلاس سپرده شده و خود کلاس repository به طور مستقیم با دیتابیس در ارتباط نیست.

حالا سراغ فایل task-repository.js میریم و کدهای زیر رو توی این فایل درج می‌کنیم.

class TaskRepository {

constructor(context) {// سازنده تابع

this.context = context;

}

createTable() { // ایجاد جدول

const sqlQuery = `CREATE TABLE IF NOT EXISTS Task (

Id INTEGER PRIMARY KEY AUTOINCREMENT,Name TEXT,Description TEXT,

IsComplete INTEGER DEFAULT 0,ProjectId INTEGER,

CONSTRAINT Task\_fk\_ProjectId FOREIGN KEY (ProjectId)

REFERENCES Project(id) ON UPDATE CASCADE ON DELETE CASCADE)`;

return this.context.run(sqlQuery);

}

insert(name, description, isComplete, projectId) { //درج

return this.context.run(

`INSERT INTO Task (Name, Description, IsComplete, ProjectId)

VALUES (?, ?, ?, ?)`,[name, description, isComplete, projectId]);

}

update(task) { // بروزرسانی

const { id, name, description, isComplete, projectId } = task;

return this.context.run(`UPDATE Task SET name = ?,Description = ?,

IsComplete = ?,ProjectId = ?WHERE Id = ?`,

name, description, isComplete, projectId, id]);

}

delete(id) { // حذف

return this.context.run(`DELETE FROM Task WHERE Id = ?`, [id]);

}

getById(id) { // گرفتن اطلاعات با ای دی

return this.context.get(`SELECT \* FROM Task WHERE Id = ?`, [id]);

}

getByProjectId(projectId) {//گرفتن اطلاعات با شناسه پروژه

return this.context.all(`SELECT \* FROM Task WHERE ProjectId = ?`, projectId);

}

getAll() {//گرفتن تمام اطلاعات

return this.context.all('SELECT \* FROM Task');

}

}

module.exports = TaskRepository;

**تنظیمات اولیه**

تقریبا کارمون رو به اتمامه. برای ساخت دیتابیس و پر کردن اون با مقادیر اولیه دو فایل config.js و seed.js رو توی پوشه‌ی context ایجاد کنید. اول سراغ فایل config.js میریم. این فایل وظیفه ساخت جداول رو داره.

//Import File

const Context = require("./context");

const ProjectRepository = require("../repository/project-repository");

const TaskRepository = require("../repository/task-repository");

//init class

const context = new Context("./database.sqlite3");

const projectRepo = new ProjectRepository(context);

const taskRepo = new TaskRepository(context);

module.exports.createTable = () => {//ساخت جداول

return new Promise((resolve, reject) => {

projectRepo

.createTable()

.then(() => taskRepo.createTable();)

.then(() => resolve("Create Table"))

.catch(err => reject("Create Table Error" + err));

});

};

با استفاده از Promise کردن تابع‌های sqlite3 توی فایل context اینجا به راحتی میتونیم زنجیره‌ای از promiseها بسازیم و از جهنم call backهای تو در تو فرار کنیم.

کدهای زیر توی فایل seed.js کپی کنید.

//Import File

const Context = require("./context");

const ProjectRepository = require("../repository/project-repository");

const TaskRepository = require("../repository/task-repository");

//init class

const context = new Context("./database.sqlite3");

const projectRepo = new ProjectRepository(context);

const taskRepo = new TaskRepository(context);

const blogProjectData = { name: "First Project" };

let projectId;

module.exports.fillData = () => {

return new Promise((resolve, reject) => {

projectRepo

.insert(blogProjectData.name)

.then(data => {projectId = data.id;

const tasks = [

{name: "First Task", description: "Start",isComplete: 1,projectId},

{name: "Second Task",description: "End",isComplete: 0,projectId}

];

return Promise.all(

tasks.map(task => {

const { name, description, isComplete, projectId } = task;

return taskRepo.insert(name, description, isComplete, projectId);

})

); })

.then(() => resolve("Fill Data is successed"))

.catch(err => {reject(err);});

});

};

**اجرای برنامه**

توی root پروژه فایل app.js رو ایجاد کنید و کدهای زیر رو داخل این فایل قرار دهید.

const config = require("./context/config");

const seed = require("./context/seed");

function app() {

config

.createTable()

.then(resolve => {console.log(resolve);

return seed.fillData();})

.then(resolve => console.log(resolve))

.catch(err => console.log(err));

}

app();

با دستور node app.js میتونید پروژه رو اجرا کنید و خروجی رو ببینید. کدهای پروژه رو از این [آدرس](https://github.com/HoseinMansouri/node-sqlite-tutorial)می‌تونید دریافت کنید. همچنین برای نوشتن این مقاله از برخی از کدهای این [مقاله](https://stackabuse.com/a-sqlite-tutorial-with-node-js/)استفاده شده.

با تشکر

حسین منصوری