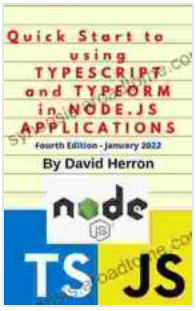
Quick Start to Using TypeScript and TypeORM on Node.js for CLI and Web

TypeScript and TypeORM are two of the most popular tools for building robust Node.js applications. TypeScript is a superset of JavaScript that adds static typing, while TypeORM is an object-relational mapper (ORM) that makes it easy to work with databases.



Quick Start to using Typescript and TypeORM on Node.js for CLI and web applications by David Herron

★★★★★ 5 out of 5

Language	: English
File size	: 3258 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 444 pages
Lending	: Enabled

In this comprehensive guide, we'll show you how to use TypeScript and TypeORM to build both CLI and web-based Node.js applications. We'll cover everything from setup and configuration to advanced topics like dependency injection and testing.

Prerequisites

To follow along with this guide, you'll need the following:

- Node.js installed on your system

- A code editor or IDE, such as Visual Studio Code
- A basic understanding of JavaScript

Getting Started

Let's start by creating a new Node.js project. Open your terminal and run the following command:

mkdir my-project cd my-project npm init -y

This will create a new directory called "my-project" and initialize a new npm project.

Next, we need to install TypeScript and TypeORM. Run the following command:

npm install --save-dev typescript typeorm

This will install TypeScript and TypeORM as development dependencies.

Now, we need to configure TypeScript. Create a file called "tsconfig.json" in the root of your project directory. Add the following contents to the file:

{ "compilerOptions": { "target": "es5", "module": "commonjs", "outDir": "./dist", "sourceMap": true, "strict": true, "noImplicitAny": true, }}

This configuration tells TypeScript to compile our code to ES5, use the CommonJS module system, and output the compiled code to the "dist" directory. We've also enabled source maps, strict mode, and implicit any type checking.

Finally, we need to configure TypeORM. Create a file called "ormconfig.json" in the root of your project directory. Add the following contents to the file:

{ "type": "mysql", "host": "localhost", "port": 3306, "username": "root", "password": "", "database": "my_database", }

This configuration tells TypeORM to use MySQL as our database, and specifies the host, port, username, password, and database name.

Building a CLI Application

Now that we have TypeScript and TypeORM configured, let's build a simple CLI application. Create a file called "index.ts" in the root of your project directory. Add the following contents to the file:

import {createConnection}from "typeorm"; import {User}from "./entity/User";

createConnection().then(async connection => { const user = new User(); user.firstName ="John"; user.lastName ="Doe"; await connection.manager.save(user); console.log("User created successfully!"); });

This script imports the "createConnection" function from TypeORM, and the "User" class from a file called "entity/User.ts". We'll create that file in a moment.

The "createConnection" function establishes a connection to our database. Once the connection is established, we create a new "User" instance and set its first name and last name. We then use the "save" method to save the user to the database.

To create the "entity/User.ts" file, run the following command:

```
touch entity/User.ts
```

Add the following contents to the file:

```
import {Entity, PrimaryGeneratedColumn, Column}from "typeorm";

@Entity() export class User {

@PrimaryGeneratedColumn() id: number;

@Column() firstName: string;

@Column() lastName: string;

}
```

This script defines the "User" entity. The "@Entity()" decorator tells TypeORM that this class represents a database table. The "@PrimaryGeneratedColumn()" decorator tells TypeORM that the "id" property is the primary key of the table, and that it should be auto-generated. The "@Column()" decorator tells TypeORM that the "firstName" and "lastName" properties are columns in the table.

Now, we can run our CLI application. Open your terminal and run the following command:

```
tsc && node dist/index.js
```

This will compile our TypeScript code and then run the compiled JavaScript code. You should see the following output:

User created successfully!

This means that our CLI application has successfully created a new user in our database.

Building a Web Application

Now that we've seen how to use TypeScript and TypeORM to build a CLI application, let's see how to use them to build a web application.

Create a new directory called "web" in the root of your project directory. This directory will contain the code for our web application.

In the "web" directory, create a file called "server.ts". Add the following contents to the file:

import express from "express"; import {createConnection}from "typeorm"; import {User}from "../entity/User";

const app = express(); app.use(express.json());

createConnection().then(async connection => { app.get("/users", async (req, res) => { const users = await connection.manager.find(User); res.json(users); });

app.post("/users", async (req, res) => { const user = new User(); user.firstName = req.body.firstName; user.lastName = req.body.lastName; await connection.manager.save(user); res.json(user); });

app.listen(3000, () => { console.log("Server is listening on port 3000"); }); });

This script imports the "express" module, the "createConnection" function from TypeORM, and the "User" class from the "../entity/User.ts" file.

The "createConnection" function establishes a connection to our database. Once the connection is established, we define two routes: a GET route for fetching all users, and a POST route for creating a new user.

To start our web application, open your terminal and run the following command:

tsc --watch

This will watch for changes to our TypeScript code and compile it automatically.

In another terminal window, run the following command:

cd web node server.js

This will start our web application. Open your browser and go to http://localhost:3000. You should see a list of all users in your database.

Click the "Create User" button to create a new user. Enter a first name and last name, and then click the "Create" button. You should see the new user appear in the list.

Advanced Topics

In this section, we'll cover some advanced topics related to using TypeScript and TypeORM.

Dependency Injection

Dependency injection is a design pattern that allows us to decouple our code from конкретные реализации. This makes our code more flexible and easier to test.

TypeORM supports dependency injection through its "injectable" decorator. To use dependency injection, we can add the "@Injectable()" decorator to our constructor functions. For example, the following script shows how we can inject the "connection" object into our "UserRepository":

```
import {Injectable}from "typeorm"; import {Connection}from "typeorm";

@Injectable() export class UserRepository {

private connection: Connection;

constructor(connection: Connection){this.connection = connection; }

// ...

}
```

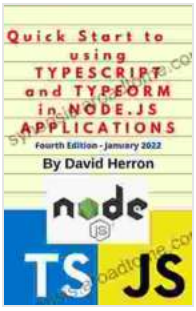
We can then inject the "UserRepository" into our other classes, such as our controllers:

```
import {Injectable}from "typeorm"; import {UserRepository}from
"./UserRepository";
```

@Injectable() export class UserController {

private userRepository: UserRepository;

constructor(userRepository: UserRepository){this.userRepository = userRepository



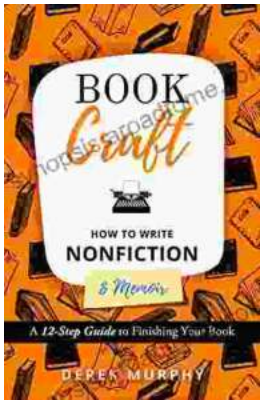
Quick Start to using Typescript and TypeORM on Node.js for CLI and web applications by David Herron

★★★★★ 5 out of 5

Language	: English
File size	: 3258 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 444 pages
Lending	: Enabled

Unveiling the Enchanting World of Customs and Crafts: Recipes and Rituals for Festivals of Light

Embark on a captivating journey through the vibrant tapestry of customs and crafts entwined with the enchanting Festivals of Light: Hanukkah, Yule, and Diwali. This...



How to Write a Nonfiction Memoir: The Bookcraft Guide

Have you ever wanted to share your story with the world? A nonfiction memoir is a powerful way to do just that. But writing a memoir can be a daunting...