TypeScript Mongo CRUD from scratch

(init)

npm -init -y
tsc --init → set dir
npm i -D nodemon, @types/express, ts-node
mongoose, mongodb, @types/mongoose
package.json → nodemon ./src/index.ts

---

set up mongodB w/ .env, connect port & dB
create mongo Schema in js
mongoose.model("User")

extend module in model type

CRUD api Tutorial

server.js file

Mongo, create deployment / project
Saved credentials in txt file
Watch guide for MongoDb set up

npm install mongodb
Const mongoose = require("mongoose")

mongoose.connect("connection url")
☆ find secure way of doing this

Connect .then(server listen).catch(error)

Model → stores items into dB
product = new mongoose.Schema({...
object of Schema (json like)
timestamps created at / updated at

Const Product = mongoose.model("Product", productSchema)

What client sends is in req.body
Send in Body in Postman for Post
Config middleware for this express.json()

API creation

Create

Save data from post into product model instance
now, on post a product is created and then res json
Post w/ data and
created at, updated at, __v → put in dB

Read API

get all entries w/ async await find {}
return all products
Find specific by id use query param :id

Update (PUT)

find product by :id using req.params
pass id and send new body (req.body)
recheck from dB

Structure

index: express, app, mongoose, mongoose URI
require (routes)
app.use (json(), urlencoded false, route includes)
mongoose connect → server listen

routes: express.Rater(), include controller functions
router.route, export router

controller: include model type from mongoose, functions,
module.exports functions.

TypeScript

Compiled and easier to test

ts → compiler → js transpilation

puts a matching js transpiled file

Configure the compiler TSC --init

target the version of JS that is being targetted

rootdir → current folder

outdir → where js will land dist (creatable folder)

Debugging launch.json and vdk lol

Types any, arrays, tuples, ...

(JS) number, string, bool, null, undefined, object

(TS) any, unknown, never, enum, tuple

any is generic type

Sometimes there is no way to explicitly type something, and as a result, you use the any type

empty declaration benefits from ts annotation of type

tuples → plain js array key-value pairs

param type and return type

No Used param / locals / returns in config

? variable → potentially used param

Objects: declare key and related type pair

Type allows for custom types

Union type allows for 2 diff input data types

Express w/ Type Setup

Dev w/ typescript → deploy JS
use import instead of require
need type files for express packages
☆☆ npm i -D @types/express ☆ for type casting

tsc --init configure compiler
noImplicitAny
strictNullChecks

install ts-node
Configure script in package "nodemon .ts"

Type Annotations

ts does type inferencing req: Request

Router: export default router
Controllers getUsers(types) import from express
include fields from intellisense
DTO → data transfer object (Models)

Type Annotating express-serve-static-core

Data Transfer Object / Interface / Class
* Right Click → type definition
Generic is like a template in C++
ResBody, ReqBody

Request<{}{}{} CreateUserDTO> EXTEND
req body now ~~extends~~ w/ this dto
query Param
{param}{RES}{REQ}{Query}

Response {ResBody}
using a dB type schema w/ resbody
Send back object

interface ExtendedParams extends Request {
  params: {}
}

Task Manager Type/Node/Mongo

get folder mit and w/ git
stored in json
Free cloud hosting w/ atlas → config URI
Collections are rows of documents
create CRUD

Put: Update resource completely
Patch: update part

Middleware to avoid try/catch block
error handler