npm add @types/nomeBiblioteca [adiciona a tipagem das bibliotecas]

npm add @types/express -D

npm add typescript [instala a dependência do typescript, para que os arquivos .ts possam ser executados pelo node]

npm tsc --init [inicializa o typescript]

npm tsc [cria o executável .js a ser executado pelo node]

\*sugere-se a modificação do tsconfig.json em [“outDir”: “/”] para a pasta que você deseja que sejam criados e alocados os .js

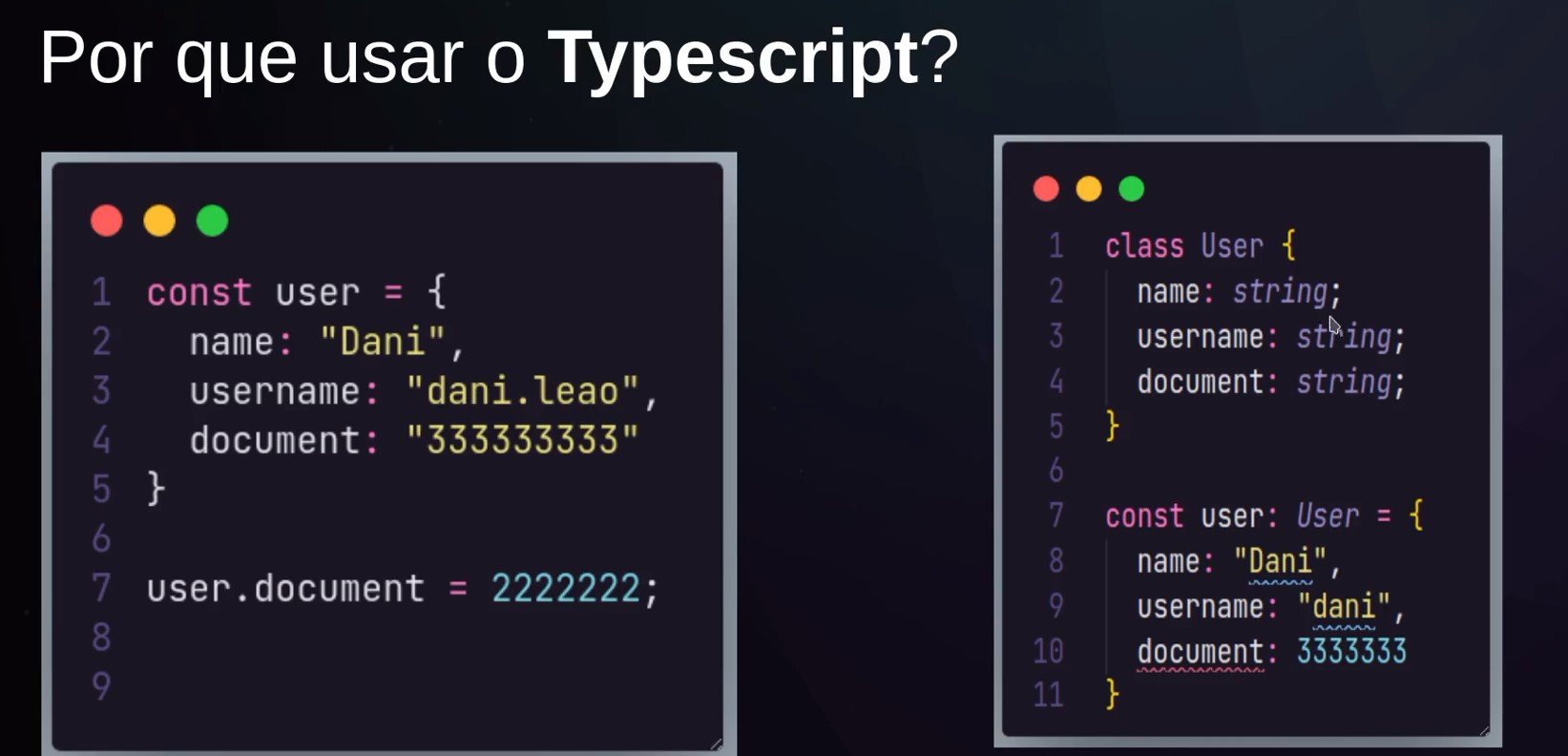
OU

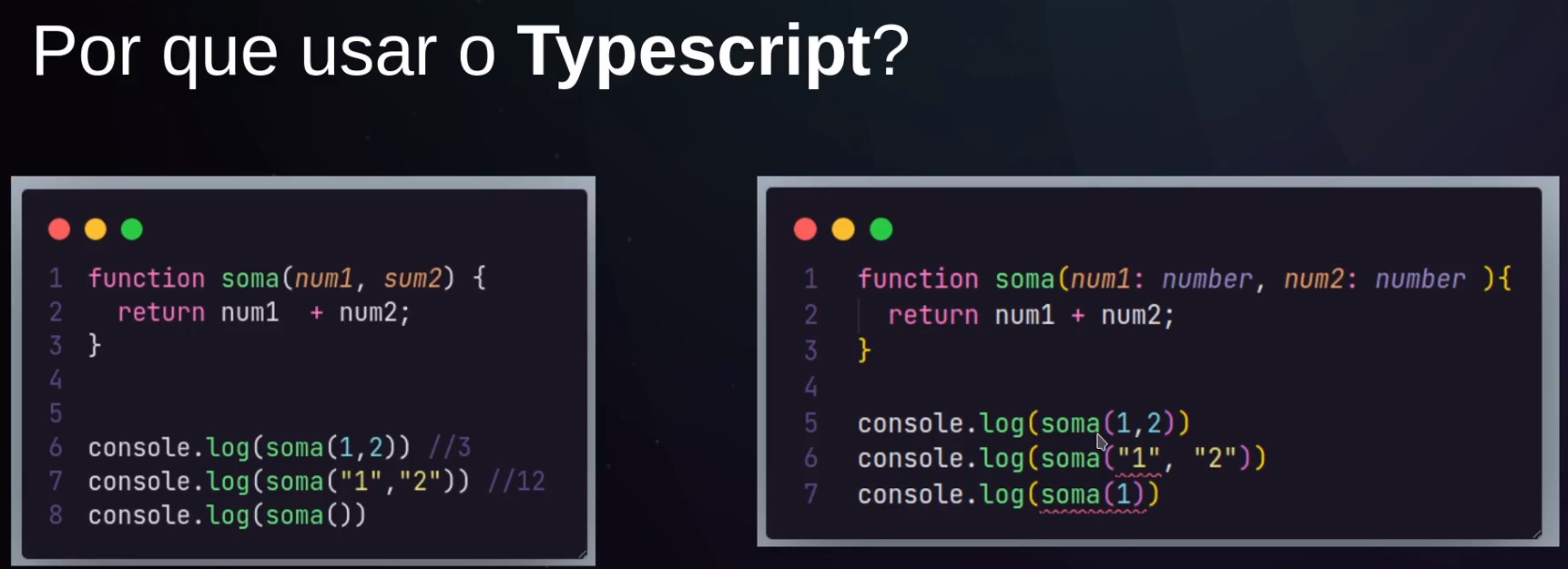
npm add ts-node-dev -D

"scripts": {

    "dev": "ts-node-dev --transpile-only --ignore-watch node\_modules --respawn src/server.ts"

  },





class Category {

    id?: string;

    name: string;

    description: string;

    created\_at: Date;

    constructor() {

        if(!this.id) {

            this.id = uuidv4()

        }

    }

}

export default Category

constructor(){} -> Adiciona método à tipagem a ser exportada como model

const category = new Category();

UTILIZAÇÃO DE REPOSITÓRIOS

1. Na rota, são recebidos os parâmetros e dados da requisição HTTP.
2. A rota chama um método do repositório correspondente para realizar a operação desejada, como criar, atualizar, buscar ou excluir dados.
3. O repositório manipula os dados usando a lógica específica do acesso aos dados, como executar consultas em um banco de dados, fazer requisições a APIs externas, ou qualquer outra ação necessária.
4. O repositório retorna o resultado da operação para a rota, que pode então enviar uma resposta apropriada ao cliente.

import fs from "fs"

import { parse } from 'csv-parse';

class ImportCategoryUseCase {

    execute(file: Express.Multer.File): void {

        //stream permite a leitura assíncrona dos dados do arquivo, em vez de carregar todo o conteúdo do arquivo na memória de uma só vez

        const stream = fs.createReadStream(file.path)

        //converte os dados CSV em uma estrutura de dados mais manipulável

        const parseFile = parse()

        //direciona os dados lidos de um fluxo para outro fluxo

        //saída do fluxo de leitura stream para a entrada do parseFile, possibilitando que os dados lidos do arquivo sejam processados e analisados pelo parseFile

        stream.pipe(parseFile)

        parseFile.on("data", async (line) => {

            console.log(line)

        })

    }

}

export { ImportCategoryUseCase }