Riga/e	Codice	Spiegazione	Richiesta Associata (dedotta)
1	const express = require("express");	Importa il modulo (framework) Express per creare il server e gestire le route HTTP.	Setup Server
2	const morgan = require("morgan");	Importa morgan, un middleware per il logging delle richieste HTTP (utile per debugging).	Setup Server (Middleware)
3	const fs = require("fs");	Importa fs (File System), un modulo built-in di Node.js per interagire con il file system (es. leggere file).	Setup Server
4	const path = require("path");	Importa path, un modulo built-in di Node.js per manipolare percorsi di file e directory.	Setup Server
5	const app = express();	Crea un'istanza dell'applicazione Express.	Setup Server
6	const cors = require("cors");	Importa cors, un middleware per abilitare Cross-Origin Resource Sharing (necessario se il frontend è servito da un'origine diversa dall'API).	Setup Server (Middleware)
8	app.use(morgan("dev"));	Utilizza morgan in modalità "dev" per loggare le richieste HTTP sulla console in un formato conciso.	Middleware (Logging)
9	app.use(express.json());	Utilizza il middleware built-in di Express per parseificare i body delle richieste in formato JSON.	Middleware (Parsing JSON)
11	app.use(cors());	Abilita CORS per tutte le route, permettendo richieste da qualsiasi origine.	Middleware (CORS)
13	app.use(express.static(path.join(__dirname, "public")));	Serve file statici (es. HTML, CSS, JS client-side) dalla directory "public". Nota: Nella struttura fornita, i file client non sono in "public", quindi questa riga potrebbe non essere usata come previsto o servirebbe a servire altri file.	Middleware (File Statici)
15	const persone = JSON.parse(fs.readFileSync(path.join(__dirname, "data.json")));	Legge il file data.json in modo sincrono, ne fa il parsing da stringa JSON a oggetto JavaScript e lo memorizza nella costante persone.	Caricamento Dati
17	app.get("/api/persone", (req, res) => { ... });	Definisce una route HTTP GET per l'endpoint /api/persone.	Endpoint API (GET all)
20	const variabile_json = { status: "success", data: persone, };	Crea un oggetto di risposta standard che include lo stato "success" e l'array completo di persone.	Endpoint API (GET all)
21	res.json(variabile_json);	Invia la risposta in formato JSON al client.	Endpoint API (GET all)
24	app.get("/api/persone/:id", (req, res) => { ... });	Definisce una route HTTP GET per l'endpoint /api/persone/:id, dove :id è un parametro dinamico (l'ID della persona).	Endpoint API (GET by ID)
25	const id = req.params.id;	Estrae il valore del parametro id dall'URL della richiesta. Sarà una stringa.	Endpoint API (GET by ID)
26	const richiesta = persone.find(r => r.id === 5);	POTENZIALE ERRORE LOGICO: Cerca una persona nell'array persone il cui id sia esattamente 5. Non usa la variabile id estratta dall'URL. Dovrebbe essere r.id === parseInt(id).	Endpoint API (GET by ID)
27-29	if (!persone) { ... }	POTENZIALE ERRORE LOGICO: Controlla se l'array persone (l'intero dataset) non esiste. Questo controllo è probabilmente errato; dovrebbe controllare se richiesta (la persona trovata) non esiste: if (!richiesta). Se non trovata (secondo la logica errata), restituisce un errore 404.	Endpoint API (GET by ID)
31	const variabile_json = { status: "success", data: richiesta };	Crea un oggetto di risposta con la persona trovata (o undefined se non trovata, o sempre la persona con id 5 a causa dell'errore).	Endpoint API (GET by ID)
32	res.json(variabile_json);	Invia la risposta JSON.	Endpoint API (GET by ID)
36-38	app.listen(3000, () => { ... });	Avvia il server Express e lo mette in ascolto sulla porta 3000. Al successful start, logga un messaggio sulla console.	Avvio Server