

# **NodeJS**

**Express**

**Json Middleware**

**More about Express & Typescript**

**MySQL2 - Path Params, Dynamic Paths, SQL Injection**

**MongoDB - Getting Started**

# **תיאוריה:**

## **מידע מגיע מהלכמה בחלקים:**

השרת מקבל קודם כל את URL ואת headers.

ורק אחר כך - מקבל את body בחלקים (chunks)

בשלכמה מסיים לשלוח הכל - סיגナル של סיום (end)

ברירת מחדל - Express לא מתייחסת לbody:  
אלא אם כן נבקש ממנה.

לכן במקרה הוסףנו middleware של json body  
הbody יהיה undefined

handshake (url+headers) —→ start reading the body —→ end

# קריאה של body בפורמט JSON

```
import { RequestHandler } from "express";

const bodyParser: RequestHandler = (req, res, next) => {
  if (req.header("content-type") !== "application/json") {
    return next();
  }

  //handle json body
  let body = "";
  req.on("data", (chunk) => {
    body += chunk;
  });

  req.on("end", () => {
    try {
      req.body = JSON.parse(body);
      next();
    } catch (e) {
      res.status(400).json({ message: "Invalid Json" });
    }
  });
}

// default export:
export default bodyParser;
// named export:
export { bodyParser };
```



אם הלקוח לא טוען שהוא שלח JSON  
אין צורך לפענח את הbody

הלקוח טוען שהוא שלח JSON

מוסיף כל חלק שקיבלנו למשתנה הזמן body

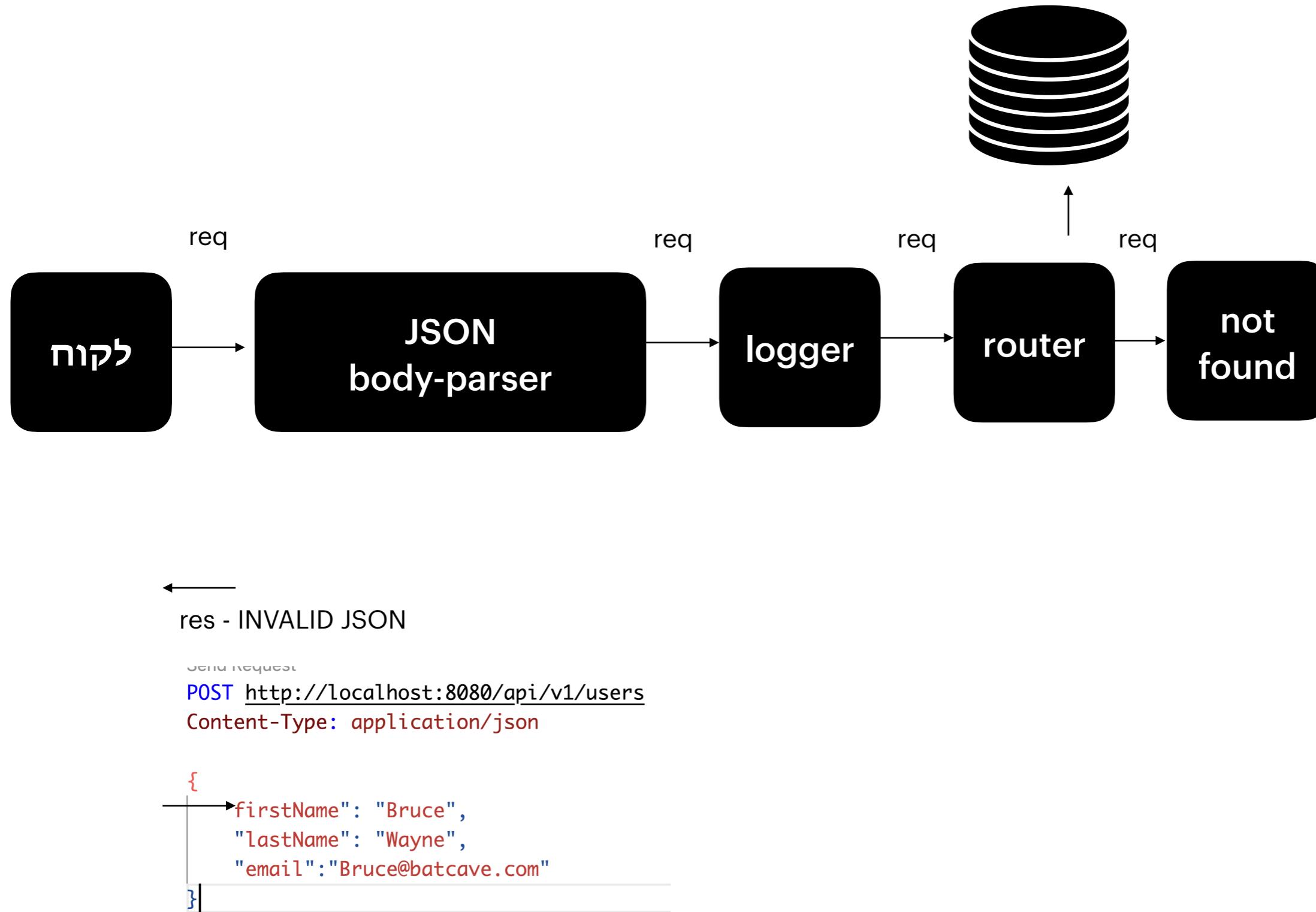
בSIGNEL סיום (אינו יותר חלקיים)  
נמיר את המחרוזת הזמן לJSON

התהליך יכול להכשל אם הJSON לא תקין  
ולבן try/catch

אם התהליך מצליח - מוסיף את הJSON  
לbody שהוא קודם undefined

אחרת נחזיר תגובה עם הסבר ללקוח  
שהJSON לא תקין.

# שרשרת של Middleware



# שימוש ב Middleware שיצרנו:

```
import express from 'express';
import { usersRouter } from './routes/users';
import { logger } from './middleware/logger';
import { notFound } from './middleware/not-found';
import bodyParser from './middleware/body-parser';

const app = express();

// middleware chain:
app.use(bodyParser);
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use(notFound);

app.listen(8080);
```

# שימוש ב Express המבנה של Middleware

```
import express, { json } from "express";
import { usersRouter } from "./routes/users";
import { logger } from "./middleware/logger";
import { notFound } from "./middleware/not-found";

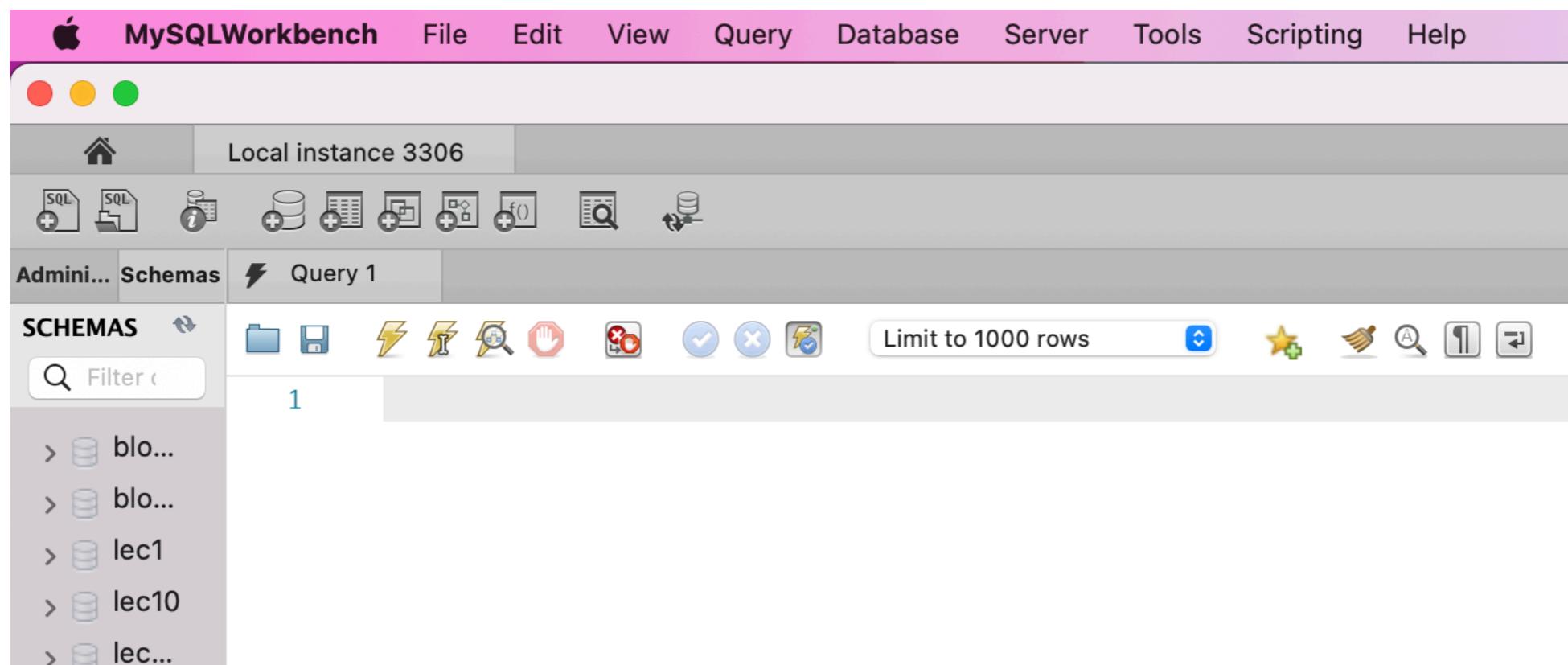
const app = express();

// middleware chain:

//json() is a function that returns a middleware function
app.use(json());
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use(notFound);

app.listen(8080);
```

# עבודה עם MySQL



# עבודה עם MySQL

```
DROP DATABASE IF EXISTS lec4;

CREATE DATABASE lec4;

USE lec4;

CREATE TABLE people(
    id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(45) NOT NULL,
    last_name VARCHAR(45) NOT NULL
);

INSERT INTO people(first_name, last_name)
VALUES ('Bruce', 'Wayne'), ('Captain', 'America');
```

# מודולווצה: יוצרים קובץ עבור הראוטר:

```
routes
  people.ts
```

```
import { Router } from "express";

const router = Router();

//router.get("/", (req, res)=>{})
//router.post("/", (req, res)=>{})

export { router as peopleRouter };
```

index.ts

```
// middleware chain:
app.use(json());
app.use(logger);
app.use("/api/v1/users", usersRouter);
app.use("/api/v1/people", peopleRouter);
app.use(notFound);
```

# עבודה עם Mysql

routes  
people.ts

<http://localhost:8080/api/v1/people>

```
import mysql2 from "mysql2";
import { Router } from "express";

const router = Router();

// TODO: move the connection to a module
const con = mysql2.createConnection({
  database: "lec4", ←
  password: "1qazxsw2", ←
  user: "root",
});

router.get("/", (req, res) => {
  con.query("SELECT * FROM people", (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.json(result);
  });
};

export { router as peopleRouter };
```

שם הדטה בweis שיצרנו  
סיסמא - להחליף לשלכם

# פונקציה חוץ עם implicit return

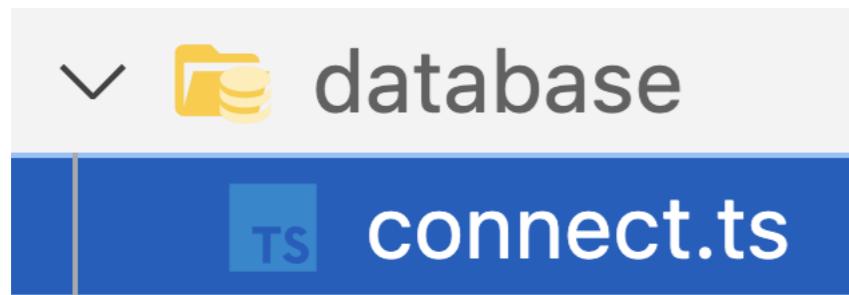
פונקציה שמחזירה ערך

```
const add2 = (x: number, y: number) => {  
    const sum = x + y;  
    return sum;  
};
```

במקרה שהפונקציה מחזירה ביטוי אחד - נשמיט את הסוגרים המסוללים  
והערך המוחזר יופיע מיד אחרי החז

```
const add = (x: number, y: number) => x + y;
```

# מודולציה: נוציא את החיבור לקובץ נפרד:



```
import mysql2 from "mysql2";

const connect = (database: string = "lec4") =>
  mysql2.createConnection({
    database: database,
    password: "1qazxsw2",
    user: "root",
  });

export default connect;
//Named export
export { connect };
```

# שימוש בפונקציה connect

```
import { Router } from "express";
import connect from "../database/connect";

const router = Router();
const con = connect("lec4");

router.get("/", (req, res) => {
  con.query("SELECT * FROM people", (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.json(result);
  });
});

export { router as peopleRouter };
```

# בקשת POST - בבקשת יש JSON Body

```
router.post("/", (req, res) => {
  const { firstName, lastName } = req.body;

  const sql = `
    INSERT INTO people(first_name, last_name)
    VALUES('${firstName}', '${lastName}')
  `;

  con.query(sql, (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.json(result);
  });
});
```

שימוש לב למכאות סביב המחרוזות

# בדיקות עם :Rest client

```
🌐 requests.http ×  
src > 🌐 requests.http > ⚒ POST /api/v1/people  
15  ### add a person:  
    Send Request  
16  POST http://localhost:8080/api/v1/people  
17  Content-Type: application/json  
18  
19  {  
20      "firstName": "Green",  
21      "lastName": "Hulk"  
22  }  
23  
24  ### get all people:  
    Send Request  
25  GET http://localhost:8080/api/v1/people
```

# שימוש בTEMPLATES Strings במקומם של placeholders

```
router.post("/", (req, res) => {
  const { firstName, lastName } = req.body;

  const sql = `

    INSERT INTO people(first_name, last_name)
    VALUES(?,?)`;

  const args = [firstName, lastName];

  con.query(sql, args, (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.json(result);
  );
});
```

1 עבור כל ערך שורצים להזירק למחוזות  
נשתמש בסימן שאלה  
VALUES(?,?)

2 נגיד מערך  
מה להכניס  
בສימן שאלה הראשון  
והשני בהתאם

3 נעביר את השאלה  
עם המערך למетодה query

# SQL Injection

```
INSERT INTO people(first_name, last_name)  
VALUES('moe', '${lastName}')
```

lastName =

```
green');DROP TABLE users; #
```

```
INSERT INTO people(first_name, last_name)  
VALUES('moe', 'green');DROP TABLE users;#)
```

```
INSERT INTO people(first_name, last_name)  
VALUES('moe', 'green');DROP TABLE users;#)
```

# שימוש ב-SQL Injection מונע placeholders

הספרייה mysql2 מטהרת את המחרוזת מכל מילת מפתח ב-SQL

DROP' במקומ 'DROP'

```
router.post("/", (req, res) => {
  const { firstName, lastName } = req.body;

  const sql = `
    INSERT INTO people(first_name, last_name)
    VALUES(?,?)`;
    ;

  const args = [firstName, lastName];

  con.query(sql, args, (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.json(result);
  });
});
```

# תוספים - הערות TODO

todo

**TODO Highlight**:  
highlight TODOs, FIXMEs, and any keywords, annotations...  
Wayou Liu

**Todo Tree**:  
Show TODO, FIXME, etc. comment tags in a tree view  
Gruntfuggly

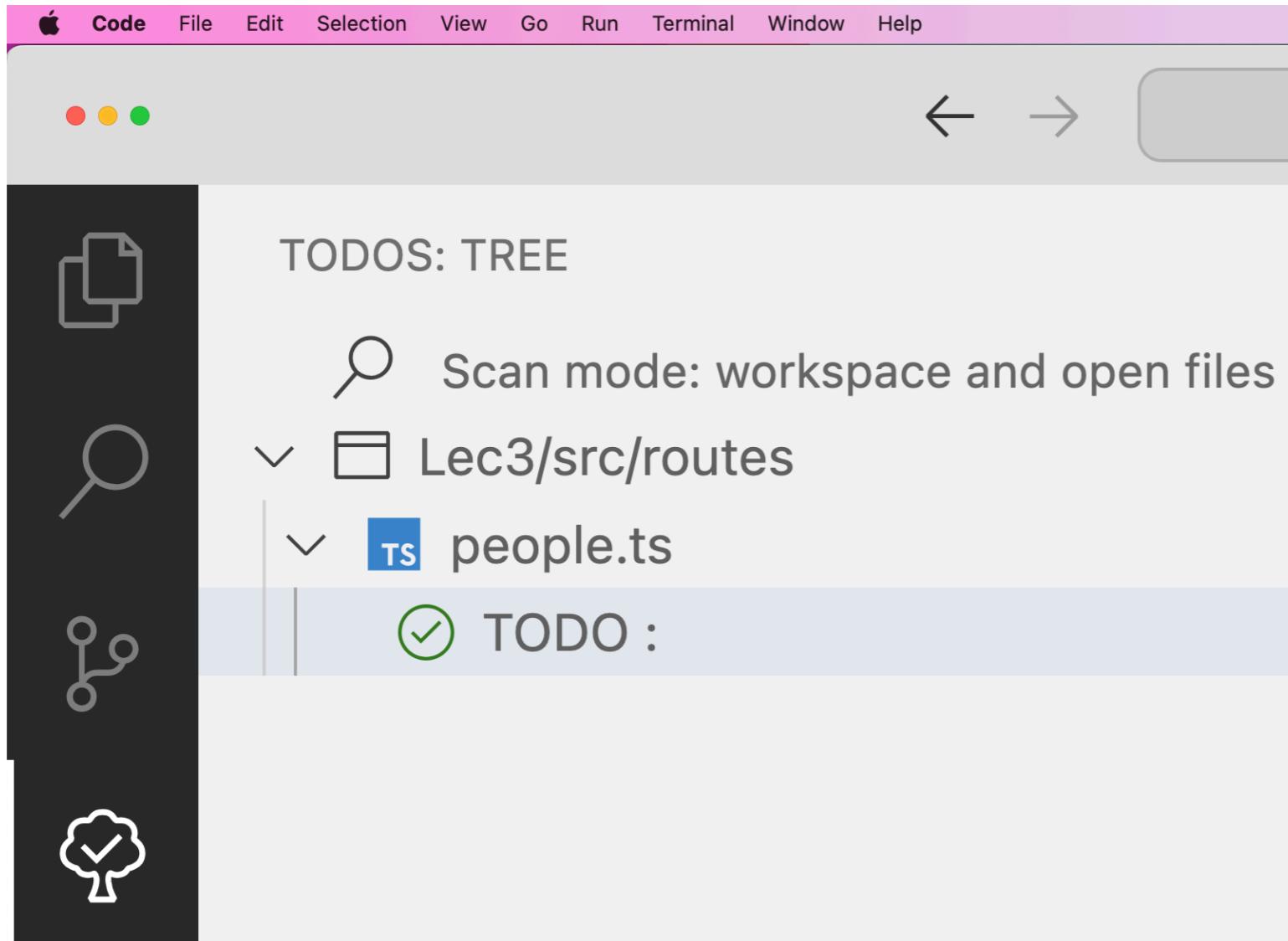
You are screen sharing Stop Share

src > routes > **TS** people.ts > ...

```
32   });
33
34 // TODO:
35 // STATUSES!
36 // REST
37 // requests with dynamic
38 // requests with query
```

עד תוסף: better comments

# הערות TODO התוסף tree



# **סטטוסים חשובים שחייבים:**

סטטוסים תקינים:

**100...199 - Informational**

**200...299 - Success**

**300...399 - redirection**

שגיאות:

**400...499 - Client Error**

**500...599 - Server Error**

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

# **סטטוסים חשובים שחייבים:**

סטטוסים תקינים:

**200 - OK**

**201 - Created**

**301 - Moved**

**400 - Bad request**

**401 - Unauthorized**

**404 - Not Found**

**500 - Internal Server Error**

# סטוס 201

בקשת POST מצינו שהבקשה הצליחה והוספנו את הערך.

```
router.post("/", (req, res) => {
  const { firstName, lastName } = req.body;

  const sql = `
    INSERT INTO people(first_name, last_name)
    VALUES(?,?)
  `;

  const args = [firstName, lastName];

  con.query(sql, args, (err, result) => {
    if (err) {
      return res.status(500).json(err);
    }
    res.status(201).json(result);
  });
});
```

# הדרות מומלצות לבניית API Rest

## Uniform interface

All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn't be too large but should contain every piece of information that the client might need.

כתובות אחידות:

במוקם להציג route עם שם ייחודי:  
/create-user

עדיף לעשות בקשה Post ל /users  
POST /users

משתמש לפי id:

/users/:id

# הדרות מומלצות לבניית API Rest

יש להפריד את צד הלוקה  
מצד השירות.

## Client-server decoupling

In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.

הלוקה נפרד מהקוד.

והלוקה פונה לשרת דרך הכתובת שלו.

# הדרות מומלצות לבניית API

**פונקציה טהורה היא פונקציה שמתבוססת רק על ערכי הפרמטרים שקיבלה כדי לחשב את התוצאה.**

## Statelessness

REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren't allowed to store any data related to a client request.

לא נרצה לנוהל Session בצד שרת.

Session שרת שומר מידע בזכרו:

לא נרצה לשמר שום מידע בזכרון גלובלי.  
במקום להשתמש במסד-נתונים.

STATE

בתכנות: מצב של אובייקט:

מצב של אובייקט - כל הערכים שלו.

תוכנות הפעולות רק על סמך הקלט הנוכחי **stateless**

- כאשר מערכת משתמשת על מצב **stateful**

<https://www.ibm.com/topics/rest-apis>

# הדרות מומלצות לבניית API Rest

1) קיבל פרמטרים ונשתמש רק בהם כדי לחשב תשובה:  
יתרון - לכל פלט - יהיה פלט צפוי.

2) קיבל פרמטרים ונשתמש גם בהם (ובערכיהם בזיכרון) כדי לחשב תשובה:  
חסרון - לא ידוע מה יהיה הפלט על סמך הפרמטרים.

```
let session = [{} , {} , {}]; →  
function savePerson(first, last) {  
    //pure function: use first and last  
    //non-pure - use memory to calculate result  
}
```

# פונקציה טהורה - לא יוצרת תופעות לוואי בזיכרון:

```
//pure function - no side effects:  
function printPerson(person: { firstName: string }) {  
  console.log(`Person: name: ${person.firstName} 😊`);  
}  
  
const bob = {firstName: "Bob"}  
printPerson(bob);
```

פונקציה טהורה:

עבור קלט נתון - נקבל תמיד את אותו פלט

# פונקציה לא טהורה - יוצרת תופעות לוואי בזיכרון:

```
function printPersonYack(person) {  
    person.firstName = `Person: name: ${person.firstName}`;  
    console.log(person.firstName);  
}  
  
const bobi = { firstName: "Bobi" };  
printPersonYack(bobi);  
printPersonYack(bobi);
```

\* פונקציה משנה  
את האובייקט\*

\* אובייקט נשמר בזיכרון

בפעם הראשונה שקוראים לפונקציה - התוצאה הגיונית.

בפעם השנייה שקוראים לפונקציה - התוצאה שונה.

**פונקציה לא טהורה:**

עבור קלט נתון - עלולים לקבל קלט שונה!

person.firstName = `Person: name: Bobi`;

person.firstName = `Person: name: Person: name: Bobi`;

**בקשו:** כתבו פונקציה שמדפיסה את המערך ממוין.

**בקשו:** כתבו פונקציה שמדפיסה את המערך שקיבלה.

```
// non-pure
function sortAndPrint(arr) {
  arr.sort();————
  console.log(arr);
}
```

אסור לשנות את האובייקט  
המיון לא תקין - 10 לא במקומות

```
// pure:
function print(arr) {
  console.log(arr);
}
```

```
// array is an object:
const arr = [4, 1, 3, 2, 0, 10];
```

```
// non-pure - can change the array!!!!!!
sortAndPrint(arr); // [0, 1, 10, 2, 3, 4]
//pure:
print(arr); // [0, 1, 10, 2, 3, 4]
```

אובייקט / מערך (סוג של אובייקט):  
אם נעביר אותו לפונקציה היא יכולה לשנות אותו.

הכתובת של האובייקט בזיכרון מועברת לפונקציה  
והfonקציה יכולה לשנות אותו בזיכרון.

(1) נכתב פונקציות טהורות.

(2) המנעוט מעובודה עם משתנים בזיכרון.

# אותו דבר עם פונקציות טהורות:

```
// pure
function sortAndPrint(arr) {
  const copy = [...arr]; ←
  console.log(copy.sort((a, b) => a - b));
}

// pure:
function print(arr: number[]) {
  console.log(arr);
}

// array is an object:
const arr = [4, 1, 3, 2, 0, 10];

// pure - can't change the array
sortAndPrint(arr); // [0, 1, 2, 3, 4, 10]
//pure:
print(arr); // [4, 1, 3, 2, 0, 10]
```

יכרנו עותק - וכך לא נשנה את המערך המקורי

כך נמיין מערך של מספרים

# הדרות מומלצות לבניית API Rest

## Layered system architecture

In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

לא לכתוב את כל הקוד במקום אחד - מודולציה.

הפרדה לשכבות

# הדרות מומלצות לבניית API Rest

## Cacheability

When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.

שרת צריך לדוח ללקוח متى השתנה המידע.

שרת צריך לדוח ללקוח - לכמה זמן המידע תקין.

הלקוח יכול לשמור את המידע לפי מה שצוין בשרת.

דוגמא: איקונים - שומר לשנה

דוגמא: מבקשים - שומר לאפס זמן

שרת מחזיר תמונה ללקוח:  
אומר ללקוח - שומר בדפדפן לשנה.

כשהלקוח ירצה את התמונה -  
localhost:8080/images/1.png

הוא ימצא אותה שמורה - ולא יבצע בקשה נוספת.

# טבלת ASCII

## מחשיים שומרים מתרגמים אותיות למספרים

### ASCII TABLE

| Decimal | Hex | Char                   | Decimal | Hex | Char    | Decimal | Hex | Char | Decimal | Hex | Char  |
|---------|-----|------------------------|---------|-----|---------|---------|-----|------|---------|-----|-------|
| 0       | 0   | [NULL]                 | 32      | 20  | [SPACE] | 64      | 40  | @    | 96      | 60  | `     |
| 1       | 1   | [START OF HEADING]     | 33      | 21  | !       | 65      | 41  | A    | 97      | 61  | a     |
| 2       | 2   | [START OF TEXT]        | 34      | 22  | "       | 66      | 42  | B    | 98      | 62  | b     |
| 3       | 3   | [END OF TEXT]          | 35      | 23  | #       | 67      | 43  | C    | 99      | 63  | c     |
| 4       | 4   | [END OF TRANSMISSION]  | 36      | 24  | \$      | 68      | 44  | D    | 100     | 64  | d     |
| 5       | 5   | [ENQUIRY]              | 37      | 25  | %       | 69      | 45  | E    | 101     | 65  | e     |
| 6       | 6   | [ACKNOWLEDGE]          | 38      | 26  | &       | 70      | 46  | F    | 102     | 66  | f     |
| 7       | 7   | [BELL]                 | 39      | 27  | '       | 71      | 47  | G    | 103     | 67  | g     |
| 8       | 8   | [BACKSPACE]            | 40      | 28  | (       | 72      | 48  | H    | 104     | 68  | h     |
| 9       | 9   | [HORIZONTAL TAB]       | 41      | 29  | )       | 73      | 49  | I    | 105     | 69  | i     |
| 10      | A   | [LINE FEED]            | 42      | 2A  | *       | 74      | 4A  | J    | 106     | 6A  | j     |
| 11      | B   | [VERTICAL TAB]         | 43      | 2B  | +       | 75      | 4B  | K    | 107     | 6B  | k     |
| 12      | C   | [FORM FEED]            | 44      | 2C  | ,       | 76      | 4C  | L    | 108     | 6C  | l     |
| 13      | D   | [CARRIAGE RETURN]      | 45      | 2D  | -       | 77      | 4D  | M    | 109     | 6D  | m     |
| 14      | E   | [SHIFT OUT]            | 46      | 2E  | .       | 78      | 4E  | N    | 110     | 6E  | n     |
| 15      | F   | [SHIFT IN]             | 47      | 2F  | /       | 79      | 4F  | O    | 111     | 6F  | o     |
| 16      | 10  | [DATA LINK ESCAPE]     | 48      | 30  | 0       | 80      | 50  | P    | 112     | 70  | p     |
| 17      | 11  | [DEVICE CONTROL 1]     | 49      | 31  | 1       | 81      | 51  | Q    | 113     | 71  | q     |
| 18      | 12  | [DEVICE CONTROL 2]     | 50      | 32  | 2       | 82      | 52  | R    | 114     | 72  | r     |
| 19      | 13  | [DEVICE CONTROL 3]     | 51      | 33  | 3       | 83      | 53  | S    | 115     | 73  | s     |
| 20      | 14  | [DEVICE CONTROL 4]     | 52      | 34  | 4       | 84      | 54  | T    | 116     | 74  | t     |
| 21      | 15  | [NEGATIVE ACKNOWLEDGE] | 53      | 35  | 5       | 85      | 55  | U    | 117     | 75  | u     |
| 22      | 16  | [SYNCHRONOUS IDLE]     | 54      | 36  | 6       | 86      | 56  | V    | 118     | 76  | v     |
| 23      | 17  | [END OF TRANS. BLOCK]  | 55      | 37  | 7       | 87      | 57  | W    | 119     | 77  | w     |
| 24      | 18  | [CANCEL]               | 56      | 38  | 8       | 88      | 58  | X    | 120     | 78  | x     |
| 25      | 19  | [END OF MEDIUM]        | 57      | 39  | 9       | 89      | 59  | Y    | 121     | 79  | y     |
| 26      | 1A  | [SUBSTITUTE]           | 58      | 3A  | :       | 90      | 5A  | Z    | 122     | 7A  | z     |
| 27      | 1B  | [ESCAPE]               | 59      | 3B  | ;       | 91      | 5B  | [    | 123     | 7B  | {     |
| 28      | 1C  | [FILE SEPARATOR]       | 60      | 3C  | <       | 92      | 5C  | \    | 124     | 7C  |       |
| 29      | 1D  | [GROUP SEPARATOR]      | 61      | 3D  | =       | 93      | 5D  | ]    | 125     | 7D  | }     |
| 30      | 1E  | [RECORD SEPARATOR]     | 62      | 3E  | >       | 94      | 5E  | ^    | 126     | 7E  | ~     |
| 31      | 1F  | [UNIT SEPARATOR]       | 63      | 3F  | ?       | 95      | 5F  | -    | 127     | 7F  | [DEL] |

# הפונקציה `Array.sort`

מסתכלת על איברי המערך בטקסט  
מיון כמו בספר טלפונים

|      |    |
|------|----|
| ace  |    |
| avi  | 1  |
| aviv | 10 |

# מיון של איברים במערך:

```
[11,2,22,1].sort((a, b) => a - b)
```

הfonקציה sort מפעילה את פונקציית חץ עבור כל זוג.  
וכך יודעת - אם להחליף או לא.

|   |   |
|---|---|
| a | b |
| 3 | 4 |

| a | - | b | מספר שלילי | הזוג סדר               |
|---|---|---|------------|------------------------|
| a | - | b | מספר חיובי | צריך להחליף ביןיהם     |
| a | - | b | אפס        | אין צורך להחליף - זהים |

לפי עקרונות REST  
צד שרת שומר את המידע בדטה-בייס ולא בזכרון.  
השתמשו במודול של `2aqsyu` כדי לשמר ולהציג מידע.

# שיעור בית:

צרו טבלה בשם `students`  
 לכל סטודנט: שם פרטי, שם משפחה, מספר סטודנט.

צרו פרויקט Typescript חדש  
 עם קובץ ראשי בשם `index.ts`  
 ומודול Router בשם `routes/students.ts`

השרת יאפשר את הפעולות הבאות:

1) להציג את כל הסטודנטים - `GET api/v1/students`

2) לחפש סטודנטים לפי שם פרטי - `GET api/v1/students/search`  
query string

3) להוסיף סטודנט `POST /api/v1/students`  
במידה והכל תקין - הטעוס יהיה 201

בדקו את תגובת השירות בעזרת קובץ `requests.rest`  
\* למה קוראים כך לסיימת של הקובץ \*

בדקו את תגובת השירות בעזרת `postman`

שימוש לב שארנו שמורים על עקרון  
כתובות אחידות ב REST



# שיעור בית:

צרו Not Found Middleware עבור

אם הלקוח מנסה לנשא לדף שלא הוגדר יש להחזיר בתגובה אובייקט JSON

```
{message: "not found"}
```

יש לציין את הסטוס - 404

ודאו שהתגובה נשלחת עם Content-Type:Application/json של HTTP Header של  
**(כדי לראות את header - יש להסתכל בtab chrome או בpostman בnetwork tab)**  
**(אפשר לראות את header גם בתגובה של התוסף rest client)**

יש להוסיף Middleware שcribes לוג את הכתובת והמתודה של הבקשות שנכנסו.  
השתמשו בספריה chalk כדי לצבוע את הטקסטocab

(לא לשכוח להשתמש middleware המובנה של express לפענוח JSON בbody)