**🧠 סיכום מורחב – שיעורים 10–11: Node.js, Express, HTTP, REST API**

**🔧 התקנות ופקודות בסיסיות**

* **התקנת חבילה גלובלית:**  
  כדי להשתמש בכלי Node.js בכל מקום במחשב (לא רק בפרויקט מסוים), מתקינים חבילה גלובלית:
* npm install -g <package-name>

למשל: npm install -g nodemon

* **בדיקת רשימת חבילות גלובליות:**
* npm list -g --depth=0

מאפשר לראות אילו חבילות מותקנות גלובלית.

**📂 עבודה עם מערכת קבצים (fs)**

* fs היא ספרייה של Node.js לעבודה עם קבצים.
* קיימות גרסאות **סינכרוניות** (blocking) ו-**אסינכרוניות** (non-blocking):
* const fs = require('fs');
* // סינכרוני
* const data = fs.readFileSync('file.txt', 'utf8');
* // אסינכרוני
* fs.readFile('file.txt', 'utf8', (err, data) => {
* if (err) throw err;
* console.log(data);
* });

**🌐 HTTP ו־HTTPS**

* **HTTP (HyperText Transfer Protocol):** פרוטוקול לתקשורת בין לקוח (דפדפן) לשרת. אינו מוצפן.
* **HTTPS:** גרסה מאובטחת של HTTP המשתמשת בהצפנת SSL/TLS.

**🔗 URL (כתובת אינטרנט)**

* URL הוא הנתיב המלא לגישה לדף אינטרנט. דוגמה:
* https://www.example.com/about

URL כולל:

* + הפרוטוקול (http/https)
  + שם הדומיין (שרת מארח)
  + נתיב (path) אל משאב ספציפי (/about)

**📡 תקשורת: בקשה ותגובה**

* **Request (בקשה):** נשלחת מהלקוח לשרת.
* **Response (תגובה):** תשובת השרת ללקוח.
* תהליך זה מתרחש בכל לחיצה על קישור או שליחת טופס.

**⚙️ פעולות CRUD**

* **Create** – יצירה (POST)
* **Read** – קריאה (GET)
* **Update** – עדכון (PUT/PATCH)
* **Delete** – מחיקה (DELETE)

**🔌 מה זה API?**

* **API = Application Programming Interface**
* מאפשר לתוכנות לתקשר אחת עם השנייה.
* **REST API** – ממשק מבוסס פרוטוקול HTTP ופעולות CRUD.

**✉️ מבנה בקשת API:**

1. **Headers** – מידע נלווה לבקשה (כמו authentication key, סוג תוכן).
2. **Endpoint** – כתובת מדויקת בשרת לפעולה מסוימת (למשל /api/users).
3. **Method** – סוג הפעולה (GET, POST, PUT, DELETE).
4. **Params / Query** – פרטים שנשלחים בכתובת (כמו מזהה).
5. **Body** – תוכן שנשלח עם הבקשה (למשל טופס).

**🚀 Express – ספרייה לבניית שרתים**

* ספרייה ב־Node.js המאפשרת ליצור שרתים בצורה פשוטה.
* התקנה:
* npm install express

**📄 דוגמה בסיסית לשרת:**

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hello World');

});

app.listen(3000, () => console.log('Server running on port 3000'));

**🧱 Middleware**

* פונקציה שרצה לפני שהתגובה נשלחת ללקוח.
* שימושים: אימות משתמש, ניתוח נתונים, טיפול בשגיאות.
* מופעלת בעזרת app.use().

**🖼 קבצים סטטיים:**

app.use(express.static('public'));

מאפשר לשלוח קבצי HTML, CSS, תמונות וכו'.

**📦 body-parser**

* בעבר הייתה ספרייה חיצונית, כיום מובנית ב־Express.
* מאפשרת לקרוא נתונים שנשלחים בטופס או ב־JSON:

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

**🔄 Fetch – שליחת בקשות מהדפדפן**

fetch('/api/user', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ name: 'Avi', age: 30 })

})

.then(res => res.json())

.then(data => console.log(data))

.catch(err => console.error(err));

**❓ מה זה Status Codes?**

* תגובת השרת בקוד מספרי:
  + 2XX – הצלחה (למשל 200 OK)
  + 4XX – שגיאת לקוח (404 Not Found)
  + 5XX – שגיאת שרת

**📝 תרגול לשיעור:**

1. מה ההבדל בין HTTP ל־HTTPS?
2. מה התפקיד של Endpoint בבקשת API?
3. מהו Middleware ולמה הוא חשוב?
4. כתוב דוגמה ל־fetch עם שליחת שם משתמש לשרת.
5. מתי נשתמש בפונקציית express.static?
6. הסבר את ההבדל בין GET ל־POST.
7. מהו קוד סטטוס 404?
8. למה חשוב לציין Content-Type בבקשת fetch?
9. כיצד יוצרים שרת Express בסיסי?
10. הסבר את ההבדל בין סינכרוני לאסינכרוני בספריית fs.