# שיעור 11

# Express

http://

# HTTP

## Hypertext Transfer Protocol

# URL

גישה לכל דף באינטרנט תיעשה על ידי שורה אחת הנקראת URL  או נתיב.
בעזרת נתיב זה נבין כמה נקודות חשובות של התקשורת בין שרת ללקוח.
אותו URL  מורכב מכמה חלקים ולכל חלק חשיבות משלו בתקשורת.

https://www.ynet.co.il/food

Secure Protocol
אוסף כללים לתקשורת בין השרת ללקוח

App domain
שם השרת המארח

Client Request
בקשת הלקוח לדף אינטרנט מסוים

# //:http

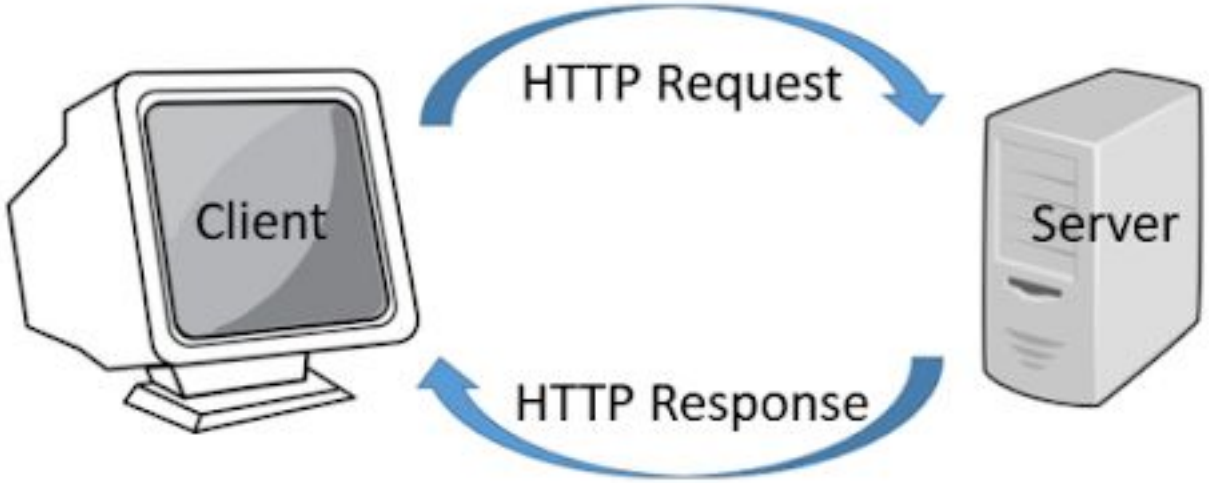
# הקדמה

מהו המושג HTTP ?

הוא פרוטוקול אינטרנט שנועד עבור העברת מידע כגון דפי html , css , javascript ואלמנטים נוספים המשמשים להצגת אתר ברשת.

התקשורת המתבצעת היא בין שתי נקודות שהן server - client ישנו דיאלוג המתקיים בין השניים הנקרא בקשה ותגובה.

לדוגמה: כאשר המשתמש לוחץ על קישור באתר, השרת מחזיר את המידע הרלוונטי והדפדפן בונה את האתר בעזרת אותו מידע.

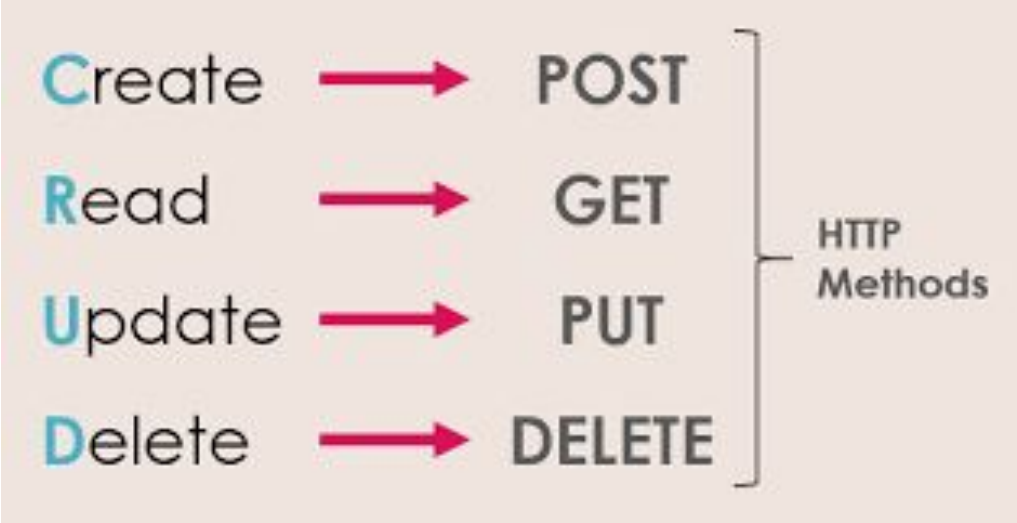


# REST API - Representational state transfer

בכדי ליצור שרת אינטרנט תחילה נבין אילו שירותים השרת מציע ואיך כל פעולה/שיטה עובדת.

ניישם מספר עקרונות עיקריים : יצירה, קריאה, עדכון ומחיקה.

עקרונות אלו נקראים CRUD operations



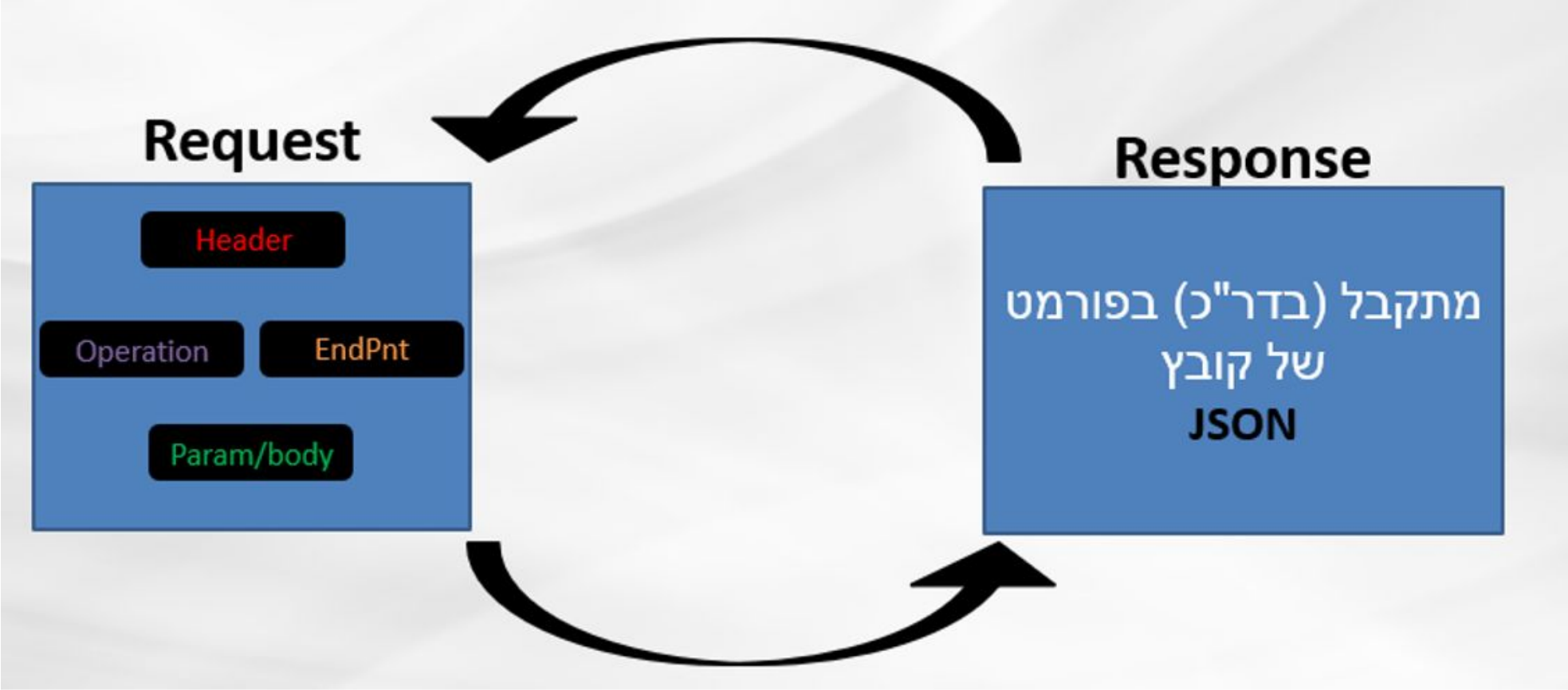
# REST API - Representational state transfer

איך נראה בקשה (Request) שנשלחת לשרת?

מורכבת מ 4 חלקים מרכזיים:

- **Header** - חלק מיוחד בבקשות API מכיל key או מידע לאימות הבקשה.
- **Operation** - מכיל אחת מפעולות הcrud.
- **Endpoint** - מיקום/ניתוב המידע השמור בתוך השרת.
- **Param/body** - מידע שנרצה לשלוח בתוך הבקשה עצמה.

# REST API - Representational state transfer



# Express Package

# Express

Express הינה ספרייה סטנדרטית ליצירת אפליקציות ווב.
ספרייה זו הינה אחת הספריות הפופולריות לבניית שרת אשר תומך באפליקציות ווב.

התקנה במסך ה CLI –

**npm install express**. יוסיף dependencies בקובץ קונפיגורציה.

```
"dependencies": {
  "express": "^4.17.1"
}
```

דוגמא לבניית שרת http אשר מאזין לפורט 3000
ומציג hello world

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => res.send('Hello World'));

app.listen(port, () => console.log(`listening to port ${port}`));
```

# Express – build server


const express = require(‘express’); הוספת הספרייה express ** אין צורך בנתיב רק בשם הספריה .

const app = express(); ייבוא האובייקט למשתנה app .

app.get (‘/’,(req,res) => res.send(‘hello world’)); הפעלת פונקצית GET מקבלת ROUT ופונקציה אשר מקבלת בקשה ותגובה.
מחזירה הודעה Hello world

*פעולתget מבקשת מידע ממקור ספציפי.

app.listen(3000 , () => console.log(‘listen to port 3000’)); האזנה של השרת לפורט ספציפי ומחזירה הודעה בconsole.log() .

# Middleware

- Middleware פונקציה אשר יש לה גישה ל respone / request של השרת.
- פונקציות אלו יכולות להכיל כל קוד, לבצע שינוים בבקשות ושליחות .
- כל middleware חייבת לסגור סיבוב או לקרוא לפונקציה הבאה.
- סיום middleware מתבצע בעזרת המתודה use אשר מוכלת בתוך express .

# Express – static files

במידה ונרצה להריץ קבצים סטטים מהשרת כמו קבצי HTML , CSS , Images וכו...'
נוכל להגדיר בעזרת הפקודה use ערוץ ספציפי שבו הקובץ יוכל לרוץ.

## server

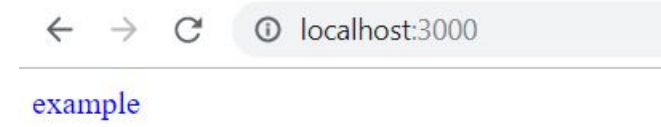
```
app.use('/',express.static('html'));
```

'/' – נתיב יחסי
'html' – שם תיקייה

## Html file

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <div id="" style="color: blue">example</div>
</body>
</html>
```

## browser



# משימה

יש ליצור3  עמודי html
העמוד הראשון יתאר את עמוד הבית.
העמוד השני יתאר עמוד רישום.
והשלישי עמוד אודות.

יש להגדיר ערוץ לכל אחד מהעמודים בנפרד.

# Express – body-parser

Body-parserהייתה ספרייה אשר נועדה לעבוד מול דפי HTML ולהעביר מידע דרך POST METHOD
את המידע ניתן להעביר דרך קבצי String , Buffer , URL , JSON .
כיום express מכילה את body-parser לכן אין צורך להתקין את הספרייה בנפרד

# Express – Form Example

בדוגמא זו אנחנו נציג דוגמא ליצירת עמוד פורום אשר יפתח בעליית השרת ולאחר הזנת הנתונים הפרטים יועברו לשרת ב . POST
לאחר מכן יועברו לערוץ אחר שם אנחנו נציג את פרטי הלקוח שנכנס.
שימו לב שform שולח את הערכים בעזרת name

```
<body>
    <form action="/submit-data" method="post">
        <label for="f-name">First Name:</label>
        <input type="text" id="f-name" name="firstName">
        <label for="l-name">Last Name:</label>
        <input type="text" id="l-name" name="lastName">
        <button>Send Data</button>
    </form>
</body>
</html>
```

יש מספר דרכים לשלוח בקשות לשרת.
נלמד אותם בהמשך, אלמנט
FORM
יודע לשלוח בקשות בעזרת
application/x-www-form-urlencoded
כדי לנתח את המידע הזה בשרת נצטרך לנתח
אותו

# Express – Form Example

```
const express = require('express')
const app = express()
const PORT = 3000

app.use(express.urlencoded({extended:false}))


app.get('/',(req,res)=>{
    res.sendFile(__dirname + '/pages/index.html')
})

app.post('/sign-up',(req,res)=>{
    const userFirstName = req.body.firstName
    res.send(`Hello : ${userFirstName}`)
})

app.listen(PORT,()=>{
    console.log(`Server works on http://localhost:${PORT}`)
})
```

משיכת הספרייה.
יצירת משתנה שמחזיק פורט
הכרזה על משיכה מה URL.

פעולת GET והחזרה של קובץ HTML.

פעולת POST מקבלת מה formאת הנתונים שהזין המשתמש , ומחזירה את שם המשתמש שקיבלה.

הפעלת השרת לפורט 3000

# Fetch

**עלינו להגדיר מספר נתונים fetch על ידי post / put / delete לצורך שליחת נתונים:**

1. בפנייה לשרת עלינו להזין 2 ערכים - הכתובת אליה נפנה והמידע שיועבר(אובייקט).

2. את המידע נעביר ברכיבי חבילת הנתונים שנשלח – headers , method , body .

3. עלינו להגדיר זאת ב, body ו-headers על מנת שנוכל לקבל/ולהעביר את הנתונים כ- json.

## ראה דוגמא בשקף הבא:

# Fetch

## post שליחת בקשה מסוג

```
fetch('/sendData', {
    headers: {
        'Accept': 'application/json',// מאפשר קבלה של json
        'Content-Type': 'application/json'// מציין את סוג התוכן שנשלח בבקשה
    },
    method: 'post',
    body: JSON.stringify({//המרת המידע שאנו נשלח - ההמרה תתבצע למחרוזת בפורמט json
        nameOfUser: userName,
        ageOfUser: userAge,
        idOfUser: userId
    })
})

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

    .then((data) => {
        // מחזיק את המידע שחזר מהשרת data
    })

    .catch((err) => {
        if (err) throw err;
    })
```

# Express – Form Example

חשוב להבין שלכל פעולה יש אירוע. אנחנו רוצים למנוע את התנהגות ברירת המחדל של אלמנט הform.
מכיוון ואנו רוצים הפעם לשלוח את הנתונים בJsonאנחנו מונעים מהאלמנט להתנהג כרגיל בכך שאנו משתמשים ,
ביכולת של()event.preventDefault
.ומשם אנו קובעים הכל. ראו דוגמא בשקף הבא

# Express – Form Example

Fetch
אנחנו נשלח בצורה הבאה:

**script**

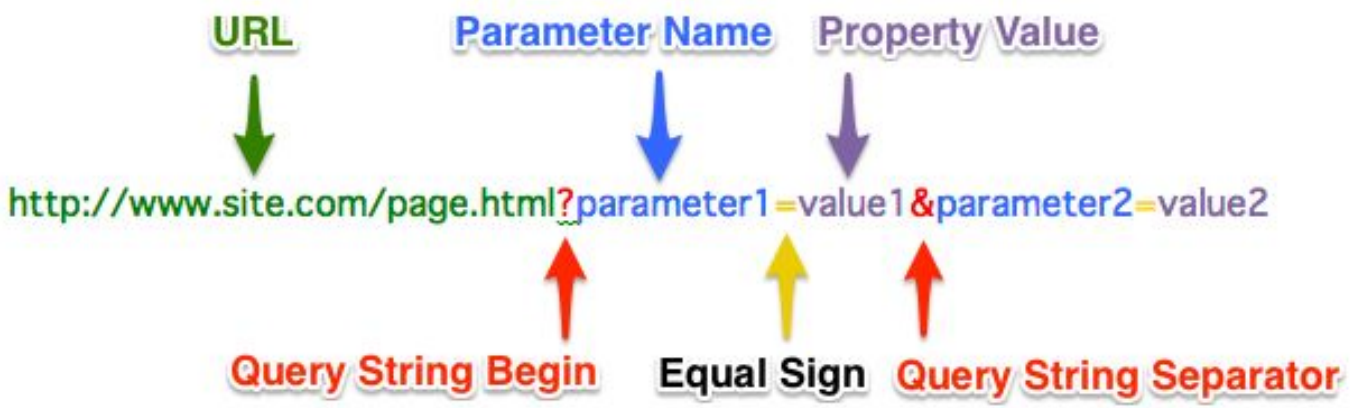
```
function sendData(event){
    event.preventDefault()
    const firstName = document.getElementById('f-name').value
    const lastName = document.getElementById('l-name').value
    fetch('/route',{
        headers:{"Content-Type":'application/json'},
        method:"POST",
        body:JSON.stringify({
            firstName,lastName
        })

    }).then(res => res.json()).then(data => {
        console.log(data)
        // כאן ניתן לעשות מה שנרצה עם התשובה מהשרת
    })
}
```

**html**

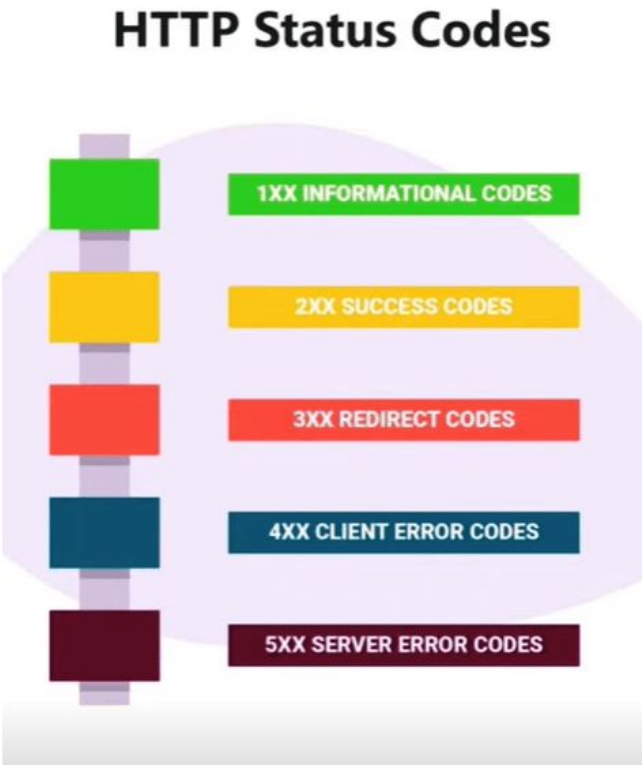
```
<form>
    <label for="f-name">First Name:</label>
    <input type="text" id="f-name" >
    <label for="l-name">Last Name:</label>
    <input type="text" id="l-name" >
    <button onclick="sendData(event)">Send Data</button>
</form>
```

# דרך נוספת לשלוח נתונים חוץ מהגוף הבקשה




ניתן לשלוח נתונים בתוך הURL.
קוראים לזה Query Parameters

# Status Codes – מה הם?




בפרוטוקול הHTTP יש לנו Status Codes , הקודים האלו בעצם הם תשובות מהשרת בקודים, האם הבקשה הצליחה? האם היה בעיה בדרך? האם אין לי אישור להיכנס לאתר הזה?

פירוט לפי סטטוסים:

1XX, למידע.

2XX , הצלחה.

3XX , העברה.

XX4 , שגיאת משתמש

5XX , שגיאת שרת

לינק לאתר בו מוסבר כל הסטטוסים

# סוף שיעור 11