הנדסת תוכנה

פיתוח צד שרת

הדר פיקאלי

תשע"ו - 2016

MEAN Stack

* בניית web applications כרוכה בשימוש בטכנולוגיות וכלים שונים, להתמודדות עם: מסד נתונים, פעולות בצד השרת, טיפול בצד הלקוח והצגה של הנתונים.
* מהו MEAN?

"MEAN is a fullstack JavaScript platform for modern web applications"

* **M**ongoDB - as the database
* **E**xpress - as the web framework
* **A**ngularJS - as the frontend framework
* **N**odeJS- as the server platform

* היום נתרכז בצד שרת NodeJS

ANGULARJS by Google

מבוא ל- NodeJS

* NodeJS היא ספריית JS שרצה על מנוע V8 של Google Chrome
* נוצר על ידי ריאן דאל בשנת 2009
* סביבת פיתוח צד שרת ב-JavaScript
* יכולה להתחבר למסדי נתונים (כמו MongoDB או MySQL)
* פועלת במערכות הפעלה Linux, Mac ו-Windows
* יכולה לרוץ דרך ה- command line

https://nodejs.org/en/

מה אפשר לעשות בעזרת NodeJS?

* שרתי צ'אט
* שימוש בספריות I/O אסינכרוניות לבניית יישומים בזמן אמת
* לבנות רשת חברתית! (LinkedIn, Dropbox)
* ועוד...

אז מה מיוחד ב- NodeJS?

* מטרתה: להוות כלי לפיתוח אפליקציות רשת שעוסקות בטיפול אינטנסיבי במידע, בזמן אמת, ושממעטות בחישובים
* במילים אחרות – הרבה I/O ומעט CPU

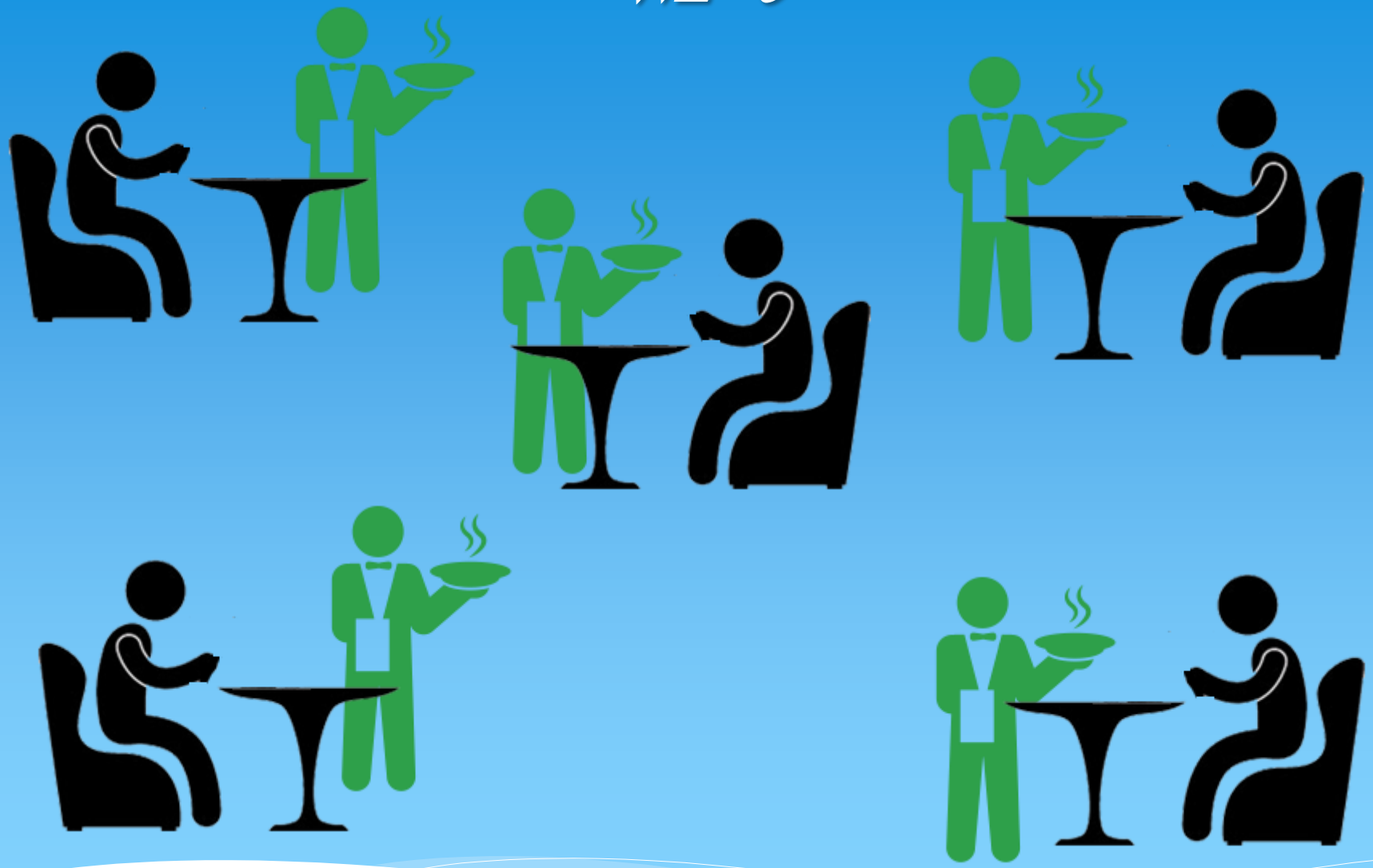
* NodeJS הוא Non- Blocking I/O
* NodeJS הוא Single-Threaded ומבוסס על מודל של אירועים Event-Loop
* הרבה מתכנתים מכירים כבר את JavaScript
* ל- Node.js יש המון מודולים ותוספות (בדומה ל- jQuery)

סיפור

node JS™

סיפור

סיפור



תכנות אסינכרוני callback

* אחת הגישות לכתיבת **קוד אסינכרוני ב- JavaScript** היא שימוש בפונקציית callback

* **JavaScript** במהות שלו אסינכרוני ותומך בפונקציה אנונימית

* כמו כן הוא אחד מהפיצ'רים החשובים של Node.js

* פונקציית callback מגדירה ערך החוזר מפעולה ממושכת. במקום להמתין לערך החוזר, ובאותו זמן לעצור את המשך ריצת האפליקציה/הסקריפט (יכול לפגוע בחוויית המשתמש), נקבל **הבטחה שבעתיד** יחזור אלינו הערך המבוקש.

כך נטפל ב- I/O בצורה טובה יותר

מתי נכון להשתמש ומתי לא?

* תלוי מה אנחנו רוצים שהפונקציה תעשה, למשל:
* אם הפעולות שנגדיר תלויות אחת בשנייה – **לא מתאים**
  * שינוי שם של קובץ לפני שינוי ההרשאות לקובץ
* אפליקציות web שצריכות כוח חישוב רציני – **לא מתאים**
* אפליקציות web שצריכות להיות סקלביליות – **מתאים**
* פעולות המתמודדות עם פעולות i/o ללא חשיבות/תלות לסדר הפעולות - **מתאים**

מתי נכון להשתמש ומתי לא?

* **Node.js בנויה מראש ל- input output מהיר אך, לא חזקה מאוד בחישוביות – לכן מתאים לאפליקציות ווב שצריכות להיות סקלביליות**

* עם זאת, אם לא נזהרים מאוד קל להיכנס לתוך חגיגת callbacks מטורפת, ב-Nodejs ניתן לטפל בזה

```
fs.chmod(oldFilename, 777, function (err) {
    fs.rename(oldFilename, newFilename, function (err) {
        fs.lstat(newFilename, function (err, stats) {
            var isSymLink = stats.isSymbolicLink();
            console.log("The file is symbolic link? "+isSymLink);
        });
    });
});
```

מתי נכון להשתמש ומתי לא?

* **Node.js בנויה מראש ל- input output מהיר אך, לא חזקה מאוד בחישוביות – לכן מתאים לאפליקציות ווב שצריכות להיות סקלביליות**

* עם זאת, אם לא נזהרים מאוד קל להיכנס לתוך חגיגת callbacks מטורפת, ב-Nodejs ניתן לטפל בזה

```
fs.chmod(oldFilename, 777, function (err) {
    fs.rename(oldFilename, newFilename, function (err) {
        fs.lstat(newFilename, function (err, stats) {
            var isSymLink = stats.isSymbolicLink();
            console.log("The file is symbolic link? "+isSymLink);
        });
    });
});
```

Blocking I/O vs. Non- Blocking I/O

הקוד הבא ישלוף את כל היוזרים מה-DB:

```
var theDiv = db.query('select * from users');
```

לעומת זאת, הקוד:

```
db.query('select * from users', function(result) {
  //do something with the result
});
```

הסבר הדוגמאות

הקוד הבא ישלוף את כל היוזרים מה-DB:

```
var theDiv = db.query('select * from users');
```

* בדוגמה הנ"ל מתבצעת שאילתא מול בסיס נתונים שתוצאותיה מיושמות במשתנה
* התהליך שמריץ את קטע הקוד הזה (בין אם זה Thread או Process או כל דבר אחר) ימתין עד קבלת תשובה חזרה מבסיס הנתונים, כאשר באותו הזמן הוא לא עושה כלום
* רק לאחר התשובה יריץ את השורות שמטפלות בתוצאה.

הסבר הדוגמאות

לעומת זאת אותה הפעולה בקוד הבא:

```
db.query('select * from users', function(result) {
  //do something with the result
});
```

* בשונה מן הדוגמא הראשונה, בדוגמה הזאת התהליך אינו ממתין ללא מעש עד לקבלת התשובה אלא משתמש בפונקציית callback אנונימית שתבוצע לאחר החזרה מבסיס הנתונים ובינתיים התהליך מתפנה לביצוע פעולות אחרות.

קוד אסינכרוני

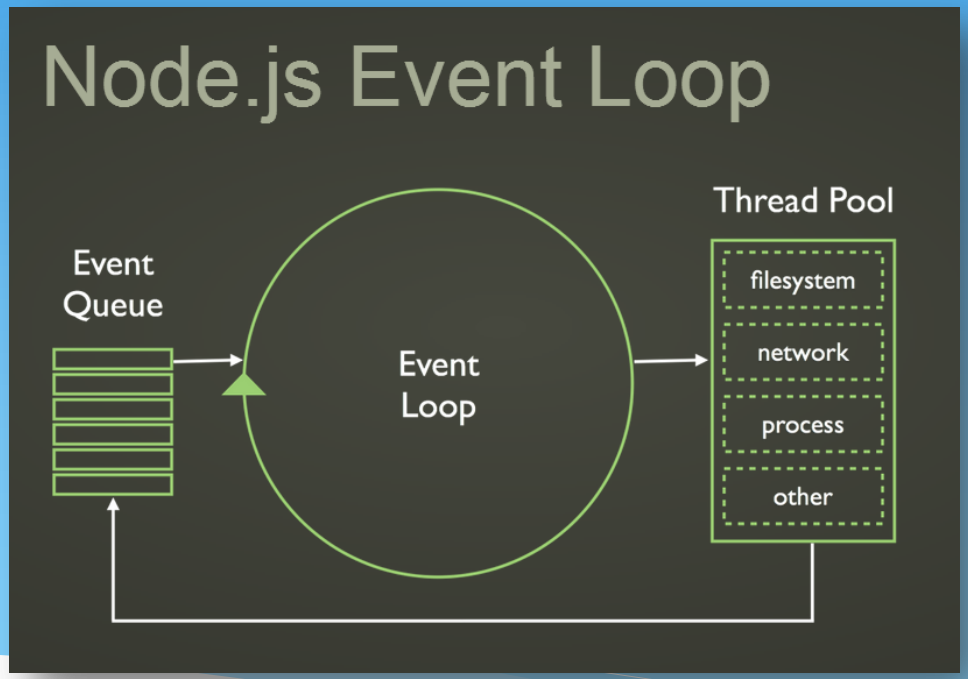
```
setTimeout( function ( )   {
    console.log( 'JCE Students‘ )
}, 2000);

console.log( 'Hello‘ );
```

JCE Students Hello **OR** Hello JCE Students

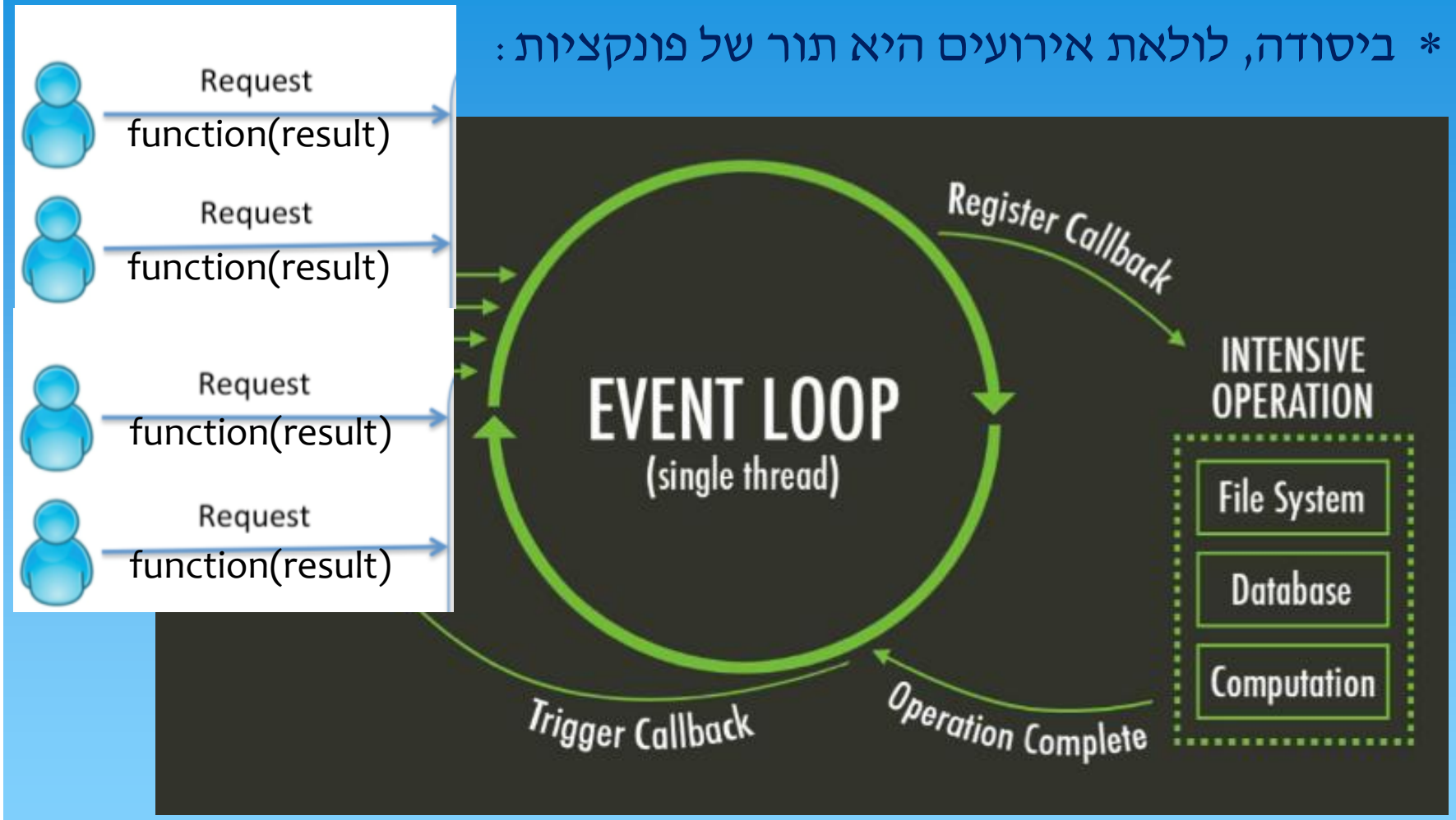
Single Threaded - Event Loop

* ה-Event Loop מאפשרת המשך הרצה, מחכה לאירועים חדשים
* כמו שראינו בדוגמא קודם, בקריאת http request נוצר אירוע ונקראת פונקציית ה-callback. Nodejs ממשיך הלאה לקריאות נוספות וכאשר ה-callback חוזר הוא מטפל בו.

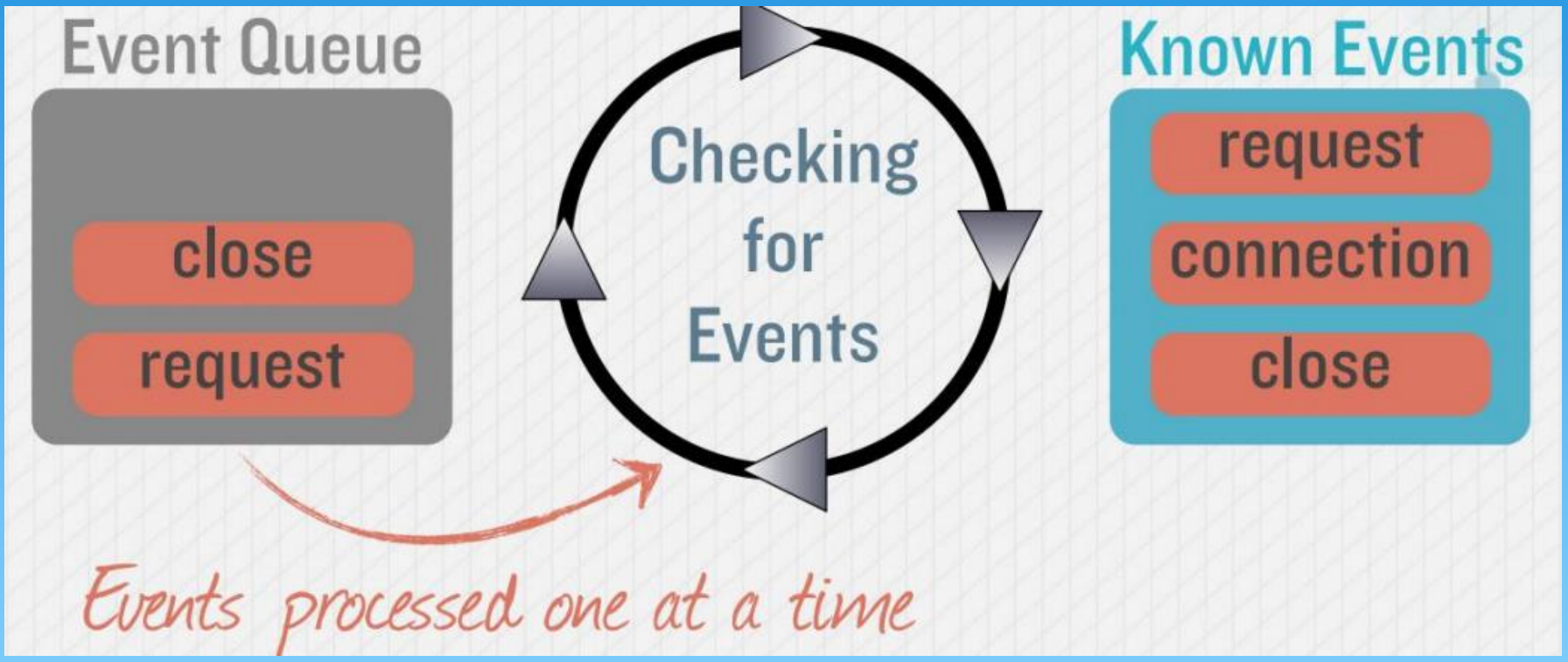


Single Threaded - Event Loop

* ביסודה, לולאת אירועים היא תור של פונקציות:



Single Threaded - Event Loop



NodeJS - Modules

* ב-NodeJS ישנם המון מודולים, המודולים הם בעצם מחלקות/ ספרייה
* המודולים מכילים (public) פונקציות מובנות שניתן להשתמש בהם
* כאשר נדרוש מודול, הוא יחזור כאובייקט JS

require('http') -> return -> HTTP object

* למשל, המודולים:
  * File System – require('fs')
  * Http – require('http')
  * Utilities – require('util')

מודול של המתכנת

Node Package Manager - NPM

* כדי לעשות שימוש בכלים שהצגנו בשקף הקודם, אנחנו צריכים להיות מסוגלים להתקין ולנהל אותם בצורה מועילה
* זוהי הנקודה בה NPM נכנס, ה- NPM מאפשר לנו להתקין את החבילות בהם נרצה להשתמש ומספק לנו ממשק לעבוד איתם
* לפני שנוכל להתחיל להשתמש NPM עלינו להתקין NodeJS (בה עם ההתקנה של NodeJS)
* ההתקנה מהלינק:

https://nodejs.org/en/download/

Node Package Manager - NPM

* איך נשתמש ב-NPM?

* find a package: search in npm.org
* install package: npm install package_name
* npm installed package in current dir: ./node_modules/
* npm install -g, install package to global node_modules dir
* לדוגמא, על מנת להתקין express נכתוב את הפקודה:

```
npm install -g angularjs --save
```

* --save add that to package.json

Package.json

* קובץ אוטומטי שנוצר
* הקובץ מכיל את כל הספריות שהשרת דורש
* בכל התקנה של ספריה עם הדגל "save--" הספרייה נרשמת בקובץ

למה צריך אותו?

* כאשר אנו מעלים את השרת לשירותים כמו גיטהאב, אנחנו לא רוצים להעלות את כל הספריות בנוסף, זה בזבוז של מקום ומיותר, לכן נעלה רק את השרת שלנו, יחד עם package.json
* כאשר מישהו מוריד את הקוד, הוא עושה npm install בתיקייה, פקודה זו תרוץ לבד על הקובץ הזה, ותתקין כל ספרייה שרשומה בו

Heroku - Package.json

* על מנת שנוכל לבצע push להירוקו, נדרש ליצור Package.json
* Package.json הוא קובץ שמכיל מידע על האפליקציה שלנו
* איך יוצרים? עם הפקודה npm init (נדרש התקנה של npm)

```
➜  heroku-node-tutorial  npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (heroku-node-tutorial)
version: (0.0.0)
description: simple node tutorial app
entry point: (server.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/coryforsyth/work/clients/itp/heroku-node-tutorial/package.json:

{
  "name": "heroku-node-tutorial",
  "version": "0.0.0",
  "description": "simple node tutorial app",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)
```

כיצד נשתמש במודול?

כיצד ניצור שרת?

איזה מודול מתאים?

דוגמא לשימוש במודול HTTP

Simple HTTP Sever

```
var http = require('http');

var server = http.createServer(function (request, response)   {

   response.writeHead(200, {"Context-Type": "text/plain"});
   response.end('Hello Wold!') ;

});
server.listen(3000);

console.log("> SERVER STARTED")
```

```
$ node server.js
```

להרצת השרת, נריץ את הפקודה
כאשר server.js הוא שם הקובץ

שימוש מערכת הקבצים

* במשך שנים, ל- JS הייתה גישה מאוד מוגבלת למערכת הקבצים. לשפת סקריפטים, גישה למערכת הקבצים נחשבת לסיכון ביטחוני מפתחים נאלצו להסתפק בעוגיות, אחסון אתרים, Flash , ActiveX וטכנולוגיות אחרות

* עם כניסתו של JS Node.js החל לצבור בסיס כשפת פיתוח צד שרת ובשרת, כניסות למערכת הקבצים הן עניין שבשגרה

NodeJS - File System Module

* אחד המודולים היותר חשובים הוא המודול שעוזר ל- NodeJS להתעסק עם קבצים ושמו: fs –file system

* FS מספק מעטפת עבור פעולות קבצים סטנדרטיות

* נקרא למודול באמצעות require

```
var fs  = require('fs');
```

NodeJS - File System Module

* לכל שיטה במודול FS יש שתי תצורות – שיטה סינכרונית ושיטה אסינכרונית

* השיטה האסינכרונית, בעלת שני פרמטרים:
  * הראשון – לשגיאות
  * השני – התוצאות (data הסופי שהתקבל)

File System Module - Open

* מבנה:

```
fs.open(path, flags[, mode], callback )
```

* **path-** הנתיב לקובץ
* **flags-** דגל המציין את ההתנהגות של הקובץ, למשל- לקריאה בלבד
* **mode-** קביעת מצב הקובץ רק במידה והוא נוצר, כברירת מחדל 0666 - לקריאה ולכתיבה
* **callback-** פונקציית ה- callback שמקבלת 2 פרמטרים (err, data)

Open File- Synchronous vs. Asynchronous

```
// Asynchronous read

var fs = require("fs");
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
    if (err) {
        return console.error(err);
    }
    console.log("File opened successfully!");
});
```

```
// Synchronous read

var fs = require('fs');
console.log("Starting");
var content = fs.open('input.txt');
console.log(" File opened successfully);
```

File System Module – Write File

* מבנה:

```
fs.writeFile(filename, data[, options], callback)
```

* **path-** הנתיב לקובץ
* **data-** המחרוזת או ה- Buffer שייכתב בקובץ
* **options**– אובייקט שיכיל {encoding, mode, flag}
כברירת מחדל,
encoding = utf8, mode = 0666, flag = 'w'
* **callback-** פונקציית ה- callback שמקבלת פרמטר אחד (err)
לשגיאה בכל מקרה של שגיאה בכתיבה לקובץ

Writing File

```
// Asynchronous read

var fs = require('fs');
console.log("Starting");
fs.writeFile(fileName.txt', "Hello World!", function(err){
        console.log("written file");
});
console.log("Finished!");
```

```
// Synchronous read

var fs = require('fs');
console.log("Starting");
fs.writeFileSync(fileName.txt', "Hello World!");
console.log("Finished!");
```

Read File - Synchronous vs. Asynchronous

```
// Asynchronous read

var fs = require('fs');
console.log("Starting");
fs.readFile('fileName.txt', function(err, data){
        console.log("Content:" + data);
});
console.log("Carry on...");
```

```
// Synchronous read

var fs = require('fs');
console.log("Starting");
var content = fs.readFileSync(fileName.txt');
console.log("Content:" + content);
console.log("Carry on...");
```

Delete File

```
// Asynchronous read

var fs = require("fs");
console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("File deleted successfully!");
});
```

```
// Synchronous read

var fs = require('fs');
console.log("Starting");
fs.unlink('input.txt');
console.log("File deleted successfully!");
```

Create a Directory

…

Writing File

Get File information

Truncate File

http://www.tutorialspoint.com/nodejs/nodejs_file_system.htm

https://nodejs.org/api/fs.html

Reading File

Closing File

…

Open a File

Remove a Directory

Create a Directory

Remove a Directory

Read a Directory

Streams

* אם הקובץ שנרצה לקרוא/ להציג למשתמש גדול וכבד מאוד – המשתמש יבהה במסך לבן

* היינו רוצים "לחלק" את הקובץ לחתיכות ואז לשגר כל חתיכה אל המשתמש על מנת שיוכל לקבל את כל הקובץ בהמשכים– כך הוא לא יבהה במסך לבן (איטיות- הזמן שלוקח לקובץ להגיע אליו, אנו טוענים את כל הקובץ לזיכרון בכל קריאה עד שהקובץ עובר בשלמותו למשתמש, אם נוכל לשבור את הקובץ לחלקים לא נצטרך לאחסן עבור כל אחד מהם את כל הקובץ בזיכרון)

* יש דרך כזו והיא נקראת stream

Streams

* Streams הם אובייקטים שמאפשרים לך לקרוא נתונים ממקור או לכתוב נתונים ליעד באופן רצוף.
* ב- Node.js ישנם ארבעה סוגים של Streams:
  * Readable- המשמש לפעולת קריאה
  * Writable- המשמש פעולת כתיבה
  * Duplex- יכול לשמש הן לשניהם- קריאה וכתיבה
  * Transform- סוג של Duplex שבו הפלט מחושבת על בסיס קלט.

https://nodejs.org/api/stream.html

MEAN Stack

* בניית web applications כרוכה בשימוש בטכנולוגיות וכלים שונים, להתמודדות עם: מסד נתונים, פעולות בצד השרת, טיפול בצד הלקוח והצגה של הנתונים.
* מהו MEAN?

"MEAN is a fullstack JavaScript platform for modern web applications"

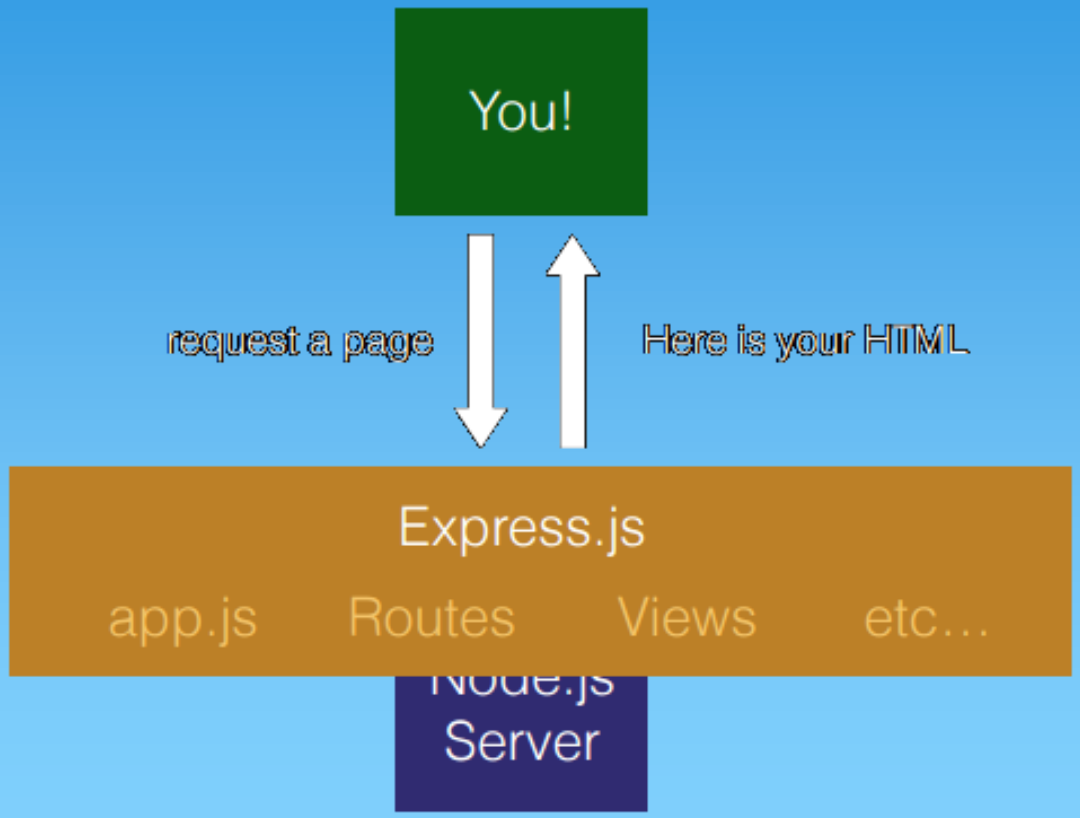
* **M**ongoDB - as the database
* **E**xpress - as the web framework
* **A**ngularJS - as the frontend framework
* **N**odeJS- as the server platform

* היום נתרכז בצד שרת NodeJS

ANGULARJS by Google

Express

A Web Framework Built On Node.js



express

מבוא ל- Express

* Express הוא Web Application Framework (מודול ב-Node) הפופולארי ביותר בשימוש עם NodeJS
* מאפשר שימוש ותכונות פיתוח פשוטות, גמישות וקלות לבניית אפליקציות web
* התקנה תתבצע כך:

```
npm install -g express --save
```

* הוספת המודול לאפליקציה:

```
var express = require('express');
var app = express();
```

הרצת הפונקציה express()

express

http://expressjs.com/en/4x/api.html

Express Middleware

* Middleware – מחברת בין רכיבי התוכנה השונים
* בעזרתו כל בקשה שמגיעה מהמשתמש תעבור דרך middleware כבר ברמת האפליקציה, לפני שהיא מגיעה לראוטינג
* בראוטינג היא גם עוברת דרך middleware אחר, לא ברמת האפליקציה אלא ברמת הראוטינג

express

Express Routing

* Express מאפשר לנו ליצור ניתוב באפליקציה
* יוצר את ה-routes וה- API
* מספק פונקציונליות 'ניתוב' בשרת HTTP

* עוזר לארגן את האפליקציה למבנה תיכון MVC
* מיני נתיבים לכל קטע, המקשר את הבקשה מהלקוח לקוד המטפל בה

express

Express Routing

* הניתוב נקבע כך-
* האפליקציה/היישום מגיב לבקשה הלקוח עבור נקודת קצה מסוימת (endpoint) שהינה URL (נתיב)

וכן לשיטה כלשהי בבקשת HTTP (GET, POST וכן הלאה)

```
app.get('/', function (req, res) {
        //.......
)};
```

יצירת ניתוב לכל בקשות ה-get

express

Express Request & Response

* Express משתמש בפונקציית callback בעלת 2 פרמטרים : **request** ו- **response**

```
app.get('/', function (req, res) {
        //.......
)};
```

* **Request-** אובייקט המייצג בקשת HTTP מהלקוח
* **Response-** אובייקט מייצג תגובת HTTP שאפליקציית express ללקוח

Routing example with Express

```
var express = require('express');
var app = express();
var path = require('path');

app.get('/', function (req, res) {
        res.sendFile(path.join(_dirname+'/index.html');
});
app.get('/about', function (req, res) {
        res.sendFile(path.join(_dirname+'/about.html');
});
```

מחזיר דף HTML של האתר
שלנו בכתובת הראשית

express

Routing example with Express

```
…..

app.get('/courses', function (req, res) {
        res.sendFile(path.join(_dirname+'/courses.html');
});
app.get('/courses/:course_Id', function (req, res) {
        var courseId = req.params.course_Id;
});
app.listen(3000);

console.log("server running at port 3000!");
```

שימוש -Dynamic URLs
בכתובות דינאמיות

האזנה לפורט
3000

express

Hello world example with Express

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
        res.send('Hello World!');
});

app.listen(3000, function () {
        console.log("server running at port 3000!");
});
```

express

Hello world – Express VS. NodeJS

```
var express = require('express');
var app = express();

app.get('/', function(request, response) {
  response.write('Hello world');
  response.end();
});

app.listen(3000);
```

using Node API

Using Nodejs API

Using Express API

```
response.send('Hello world')
```

express