
EXPRESS

Fast, unopinionated, minimalist web framework for [Node.js](#)

! יש לעבור על המציגת של [postman](#) לפני שימושיכים בציגת זאת

<http://expressjs.com/>

תוכן עניינים

1	Introduction
4	<u>listen</u>
8	<u>Express middlewares</u>
11	<u>Application-level middleware</u>
15	<u>response object</u>
21	<u>request object</u>
28	<u>Built-in middleware</u>
32	<u>Express static</u>
37	<u>Error-handling middleware</u>
40	<u>Router-level middleware</u>
52	<u>Third-party middleware</u>
53	<u>CORS</u>
64	<u>morgan</u>



Javascript in server side

Scale our application quickly

Supported by Google v8 engine

Community support

Free and open-source

Great performance

Ease of use

Rapid development

Good documentations



listen

מטרודה של express הפותחת מאזין
ליירוט בקשות <http://>



express.listen

js app.js X

EXPRESS-MIDDLEWARES > LISTEN > js app.js > ...

```
1 const express = require("express"); ←  
2 const app = express(); ←  
3 const chalk = require("chalk");  
4  
5 app.get("/", (req, res) => { ←  
|   res.send("In David Yakin app!"); ←  
7 });  
8  
9 const PORT = 8181;  
10 app.listen(PORT, () =>  
11 |   console.log(chalk.blueBright("Listening on: http://localhost:8181"))  
12 );
```

- ניבא את הפונקציה `express` לתוך קבוע בשם `express`
- נשמר את האובייקט שיחזור מהפעלת הפונקציה `express` לתוך קבוע `app`
- פועל את מетодת `get` שתייצור `end point` (מיוצג כתובות) שמקבלת שני פרמטרים:
 - Path – נתיב לירוט
 - Call back – הֆונקציה יכולה לקבל עד שלושה פרמטרים כאשר:
 - Request – אובייקט הבקשה
 - Response – אובייקט התגובה
 - Next – העברת שלושת הפרמטרים לפונקציית ה- `call back` הבא
- נשימוש באחת המethodות מתוך אובייקט התגובה `send` שתשלח את מחרוזת התווים
- פועל את מетодת `listen` שמקבלת שני פרמטרים:
 - Port – מספר בין אפס ל – 65,535
 - Call back – תדפיס בקורסול את הכתובת אליה אננו מאזינים

Express – היא פונקציה שניית להגעה לחלק מהפתחות שבתוכה ללא הפעלה !

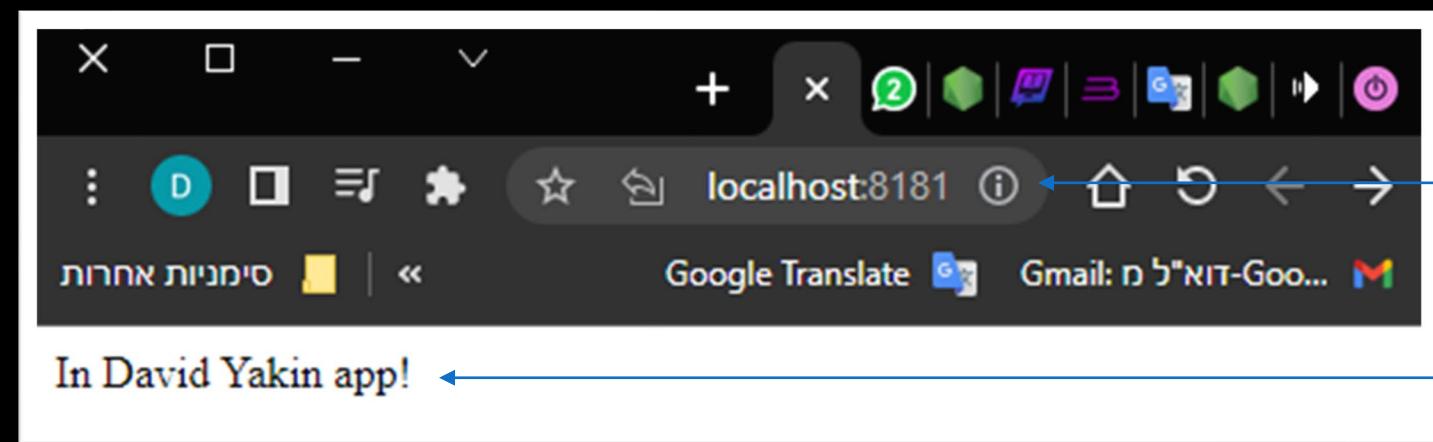
התוצאות

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

C:\Users\DELL\Desktop\HackerU\lecturer-work\Lessons\NODE.JS\NODE.JS-SANDBOX\EXPRESS-MIDDLEWARES\LISTEN>npm start
Debugger attached.

> app.listen@1.0.0 start
> node .

Debugger attached.
Listening on: http://localhost:8181 ←
```



- אם הפעלת הקוד תודפס בקונסול מחרוזת התווים שהעברנו בפונקציית ה – call back בפרמטר השני של METHOD listen

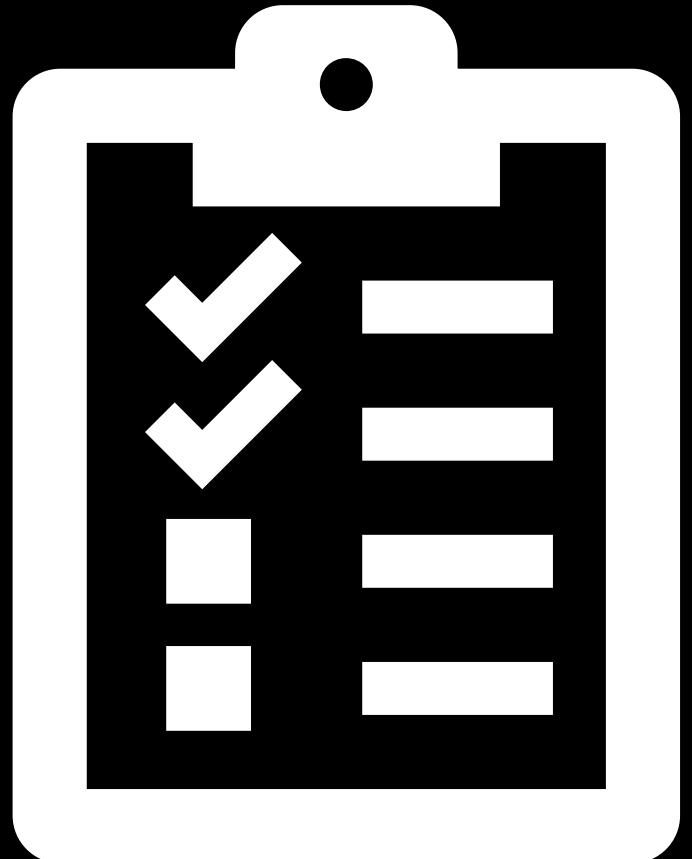
- אם נגלוש לכתובת הzzzz נקבל את הכתובת מתוך METHOD app.get ZZZZ מכיוון שהדף בכתובת הzzzz מבצע בקשת get מכתובת ה – DNS ומהפורט שציגנו

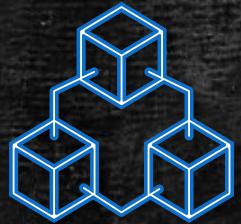
! אם ברצוננו להפסיק את המאוזן נキיש בטרמינל: Ctrl + c

Business-cards-app

- server.js - צור מאזין על פורט 8181 באמצעות מטודת listen של express תן חיובי על הצלחת התהיליך וכותב בטרמינל בצבע כחול בהיר את הכתובת אליה השרת שיצרת מאזין

משימת
express.listen





Express middlewares

functions that have access to the [request object](#) (req), the [response object](#) (res), and the next middleware function in the application's request-response cycle.



Middleware functions can perform the following tasks:

01

Execute any code.

02

Make changes to the request and the response objects.

03

End the request-response cycle.

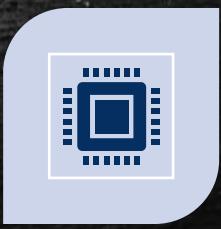
04

Call the next middleware function in the stack.

- ! If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging.

Middleware types

סוגי זה – express ו-middlewares בהם:



APPLICATION-LEVEL
MIDDLEWARE



ROUTER-LEVEL
MIDDLEWARE



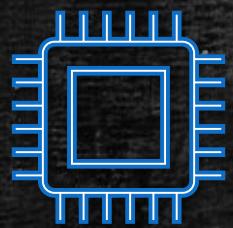
ERROR-HANDLING
MIDDLEWARE



BUILT-IN
MIDDLEWARE



THIRD-PARTY
MIDDLEWARE



Application-level middleware

פונקציות ביניים מותן ליבת express



app.use

JS app.js

```
EXPRESS-MIDDLEWARES > APP-USE > JS app.js > ...
1  const express = require("express");
2  const app = express();
3  const chalk = require("chalk");
4
5  app.use((req, res, next) => { ←
6    console.log(chalk.yellowBright("in first app.use!"));
7    next();
8  });
9
10 app.use("/", (req, res, next) => { ←
11  console.log(chalk.yellowBright("in second app.use!"));
12  next();
13});
14
15 const PORT = 8181;
16 app.listen(PORT, () =>
17  console.log(chalk.blueBright("Listening on: http://localhost:8181"))
18);
```

```
Listening on: http://localhost:8181
in first app.use!
in second app.use!
```

פונקציית middleware שיכולה
לקבל בין פרמטר אחד לאין סוף
פרמטרים

- פונקציה המוגדרת כ – middleware זאת פונקציה שמקבלת פונקציית call back עם בין שלושה עד ארבעה פרמטרים כאשר האחרון מהם הוא תמיד הפרמטר next הפעלה מוצעת next בתוך פונקציית call back תעביר את כל האובייקט של express למטרודה הבאה בדוגמה הבאה אני יוצר שני מירוצים באמצעות app.use:
 - בראשון אני מעביר בפרמטר פונקציה get post put שתיירת את כל סוגי הבקשות (patch delete) הפונקציה תדפיס בצדב את מחזורות התווים ותפעיל את מטודת next כך שהת hollow בתוכה הפונקציה יסתים והיא תחזיר את אובייקט ה express
 - בשני אני מעביר בפרמטר השני את כתובות ה – root ובפרמטר השלישי את פונקציית callback גם פונקציה זאת תדפיס בטרמינל מחזורות תווים.
- התוצאה: ניתן לראות שניות המטודות פעלו בצדקה זהה.

חשיבות response על מנת לשלוח תשובה בחזרה מהשרת לאחר האפליקציה תחקע על מצב pending !

הפעלת מספר פונקציות באמצעות `app.use`

```
15 app.use(  
16   "/", ←  
17   (req, res, next) => { ←  
18     console.log(chalk.yellowBright("one"));  
19     next();  
20   },  
21   (req, res, next) => { ←  
22     console.log(chalk.redBright("two"));  
23     next();  
24   },  
25   (req, res, next) => { ←  
26     console.log(chalk.magentaBright("three"));  
27   }  
28 );
```

```
[nodemon] restarting due to changes...  
[nodemon] starting `node .`  
Listening on: http://localhost:8181  
one  
two  
three ←  
[]
```

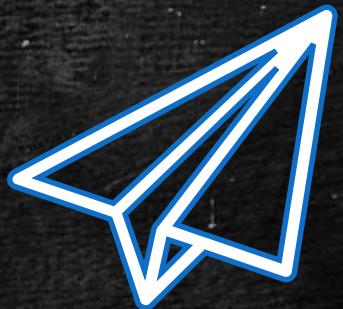
- פונקציית middleware שיכולה לקבל בין פרמטר אחד לאין סוף פרמטרים
- ניתן לשדרר כמהות אין סופית של מethodות בثانיה `use` כל עוד בסוף כל אחת מהם מפעילים את מethodת `next`
- דוגמה שהלן אני משתמש במmethod `app.use` כאשר:
- בפרמטר הראשון אני מעביר לה את הכתובת שהיא תירט את הבקשות
- בפרמטר השני פונקציה `call back` שתדייס מחרוזת תווים בצבוע צהוב בקונסול ולאחר מכן תפעיל את מmethod `next`
- בפרמטר השלישי פונקציה `call back` נוספת שתדייס שתהופיע בקונסול מחרוזת תווים בצבוע אדום ותפעיל גם היא את מmethod `next`
- בפרמטר הרביעי ענביר שוב `call back` שתדייס בקונסול מחרוזת תווים בצבוע מג'נטה
- התוצאה בטראמיניל

Middlewares

```
30 app.get("/", (req, res, next) => {
31   console.log(chalk.yellowBright("in get method!!!"));
32   next();
33 });
34
35 app.post("/", (req, res, next) => {
36   console.log(chalk.yellowBright("in post method!!!"));
37   next();
38 });
39
40 app.patch("/", (req, res, next) => {
41   console.log(chalk.yellowBright("in patch method!!!"));
42   next();
43 });
44
45 app.put("/", (req, res, next) => {
46   console.log(chalk.yellowBright("in put method!!!"));
47   next();
48 });
49
50 app.delete("/", (req, res, next) => {
51   console.log(chalk.yellowBright("in delete method!!!"));
52   next();
53 });
```

כפי שניתן לראות המethodות הבאות גם כויכולות לשימוש כ – middleware בהתנאי שיפעלנו בתוכם את מטודת next ולא יסגרו את הבקשה.

בניגוד למетодת `app.use` שמירשת את כל סוגי הבקשות, מethodות אילו יירתו בקשות רק מהסוג שהן נקראות על שמן



RESPONSE OBJECT

אובייקט התגובה

<http://expressjs.com/en/5x/api.html#res>



res.send

```
58 app.get("/", (req, res, next) => { ←  
59   |   res.send({ name: "david" });  
60 });
```

The screenshot shows the Postman interface. At the top, there's a header with 'GET' and 'localhost:8181'. Below it, under 'Body', there's a note: 'This request does not have a body'. In the 'Headers' tab, there are seven items listed. On the right side, there's a 'Cookies' section and a 'Save Response' dropdown.

Under 'Test Results', the response is displayed in JSON format:

```
1 [ {  
2   "name": "david"  
3 } ]
```

מטרה שסוגרת בבקשת HTTP ויש
ביכולתה לשלוח נתונים מצד שרת לצד
לקוד

בדוגמה שללנו :

- אני מירט בקשה מסוג get לכל כתובות ("/") ומשתמש במטרה מתוך אובייקט התגובה (הפרמטר השני תמיד) בפונקציית ה – call back בשם send כדי לשלוח תגובה מהשרת לצד لكוד עם אובייקט שיצרתني

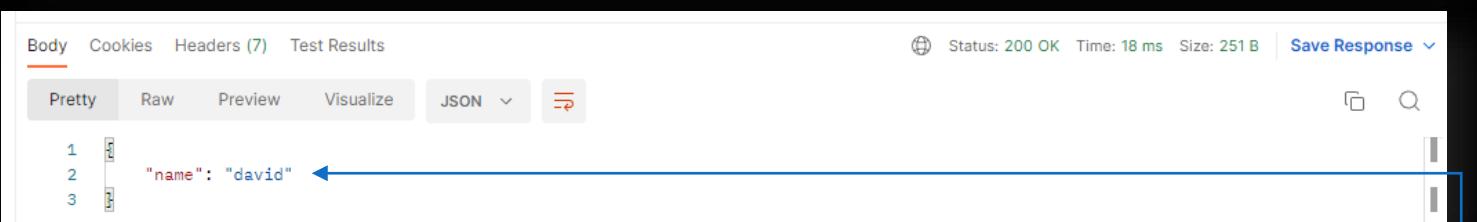
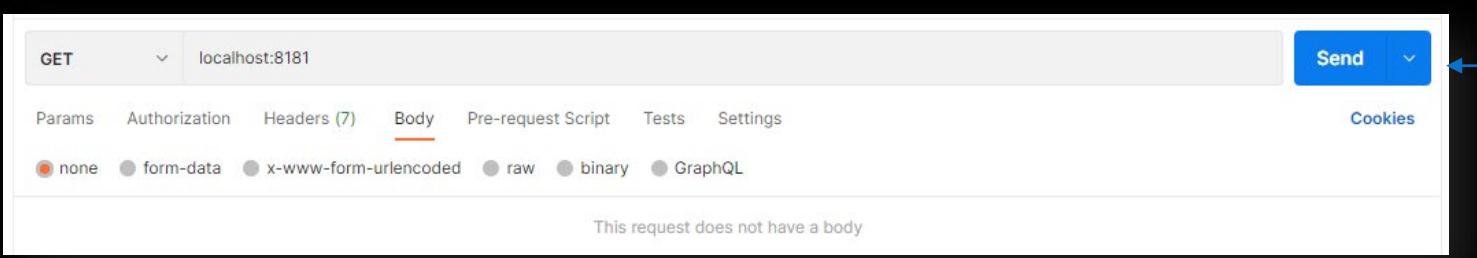
בתוכנת postman :

- אני שלוח את הבקשה
- מקבל בתשובה את האובייקט

אני יכול להשתמש במטרה res.send כדי לשלוח גם מחוצת תווים, מערך, Boolean, undefined, null אך לא מספרים.

res.send

```
62 app.get("/", (req, res, next) => {  
63   res.send({ name: "david" });  
64   setTimeout(() => console.log("in send"), 2000);  
65});
```



```
Listening on: http://localhost:8181  
in send
```

חשוב לציין כי למרות שmethod זאת חסוב לדעת כי לשליחת הבקשה אם כתוב בתוך פונקציית ה - `call back` קוד נוסף שיש לבצע הוא יבוצע גם לאחר החזרת התשובה לצד לקוח.

בדוגמה שללון :

- לאחר החזרת התשובה לצד לקוח בעזרת method `res.send` אני מפעיל את המטודה `setTimeout` שלאחר שתי שניות תדפיס בקונסול את מחזורת התווים.

בתוכנת postman :

- אני שלוח את הבקשה
- מקבל בתשובה את האובייקט
- ניתן לראות שלאחר שתי שניות מופיעה בקונסול מחזורת התווים שכתבתי

ישנה פונקציה נוספת בשם `res.json` שעשויה פחות או יותר את מה שmethod `res.send` עשויה רק היא יותר מוגבלת באפשרויות של העברת הנתונים לצד לקוח ולכן נשתמש בעיקר בmethod `res.send`

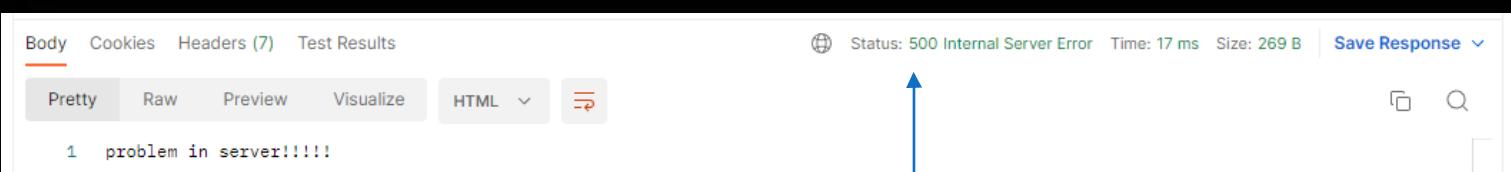
res.status

מטרודה שבעזרתה ניתן לעשות השמה
למפתח ה - `status code` באובייקט
התגובה :

בדוגמה שלהן :

- נשתמש במטרודה `res.status` שמקבלת
בפרמטר מספר
- בגל שמטרודה זאת מחזירה את
אובייקט התגובה לאחר השינויים אני
יכול לשגר לו את מטרודת `send` כדי
לשגר הודעה תשובה
- בתוכנת `postman` :
 - ניתן לראות שהערך במפתח ה - `status`
השתנה למספר שנמתי
 - ניתן לראות את הודעה התשובה
שלחתית

```
67 app.get("/", (req, res, next) => {  
68   |   res.status(500).send("problem in server!!!!"); ←  
69 });
```



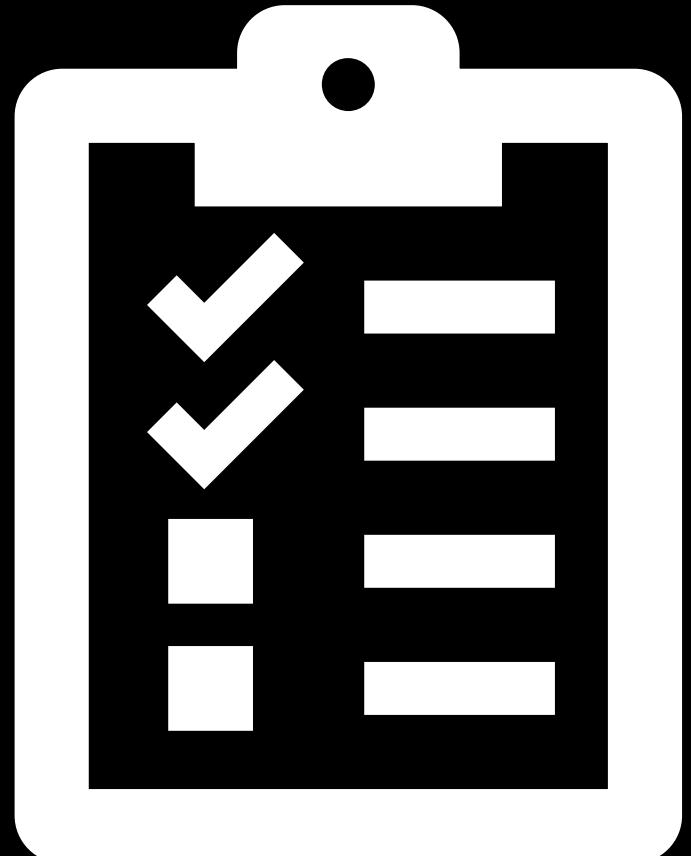
לרשימה המלאה של מספרי ה `status code` והמשמעות שלהם יש לחוץ על הקישור הבא:

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

node-sandbox

- צור תיקייה חדשה בשם sandbox-server
- צור קובץ בשם app.js
- אתחל פרויקט node.js חדש בתחום התיקייה
- הורד לפרויקט את nodemon ככלי מפתח
- הורד לפרויקט את chalk בגרסת 4.1.1
- הורד לפרויקט את express
- – package.json צור את ה – scripts שיפעלו את האפליקציה בעזרת node ו – nodemon
- בקובץ app.js צור מאזין על פורט 8181 באמצעות методת listen של express תן חיוני על הצלחת התחילה
- בטרמינל בצע צהוב בהיר את הכתובת אליה השרת שיצרת מאזין

משימת response



node-sandbox

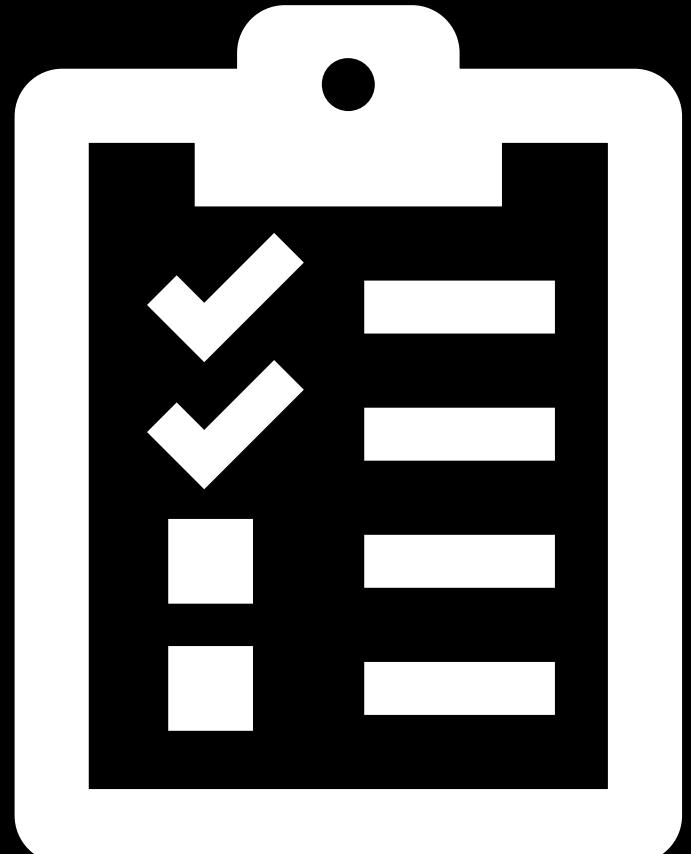
- צור את המירטטים הבאים:

Method	url	response
Get	/user	{name: user, age: 55}
Post	/	[{name: user, age: 55},{name: second, age:3}]
Delete	/1	User deleted
Put	/2	User was updated
Patch	/3	User like post

- הפעיל את הקוד שבקובץ `entry point` באתרים

הפקודה המתאימה
- בדוק באמצעות תוכנת `postman` את שליחת כל בקשה
ה – `http://localhost:3001` יוצרת מירטטים.

חלק ב'





REQUEST OBJECT

אובייקט הבקשה

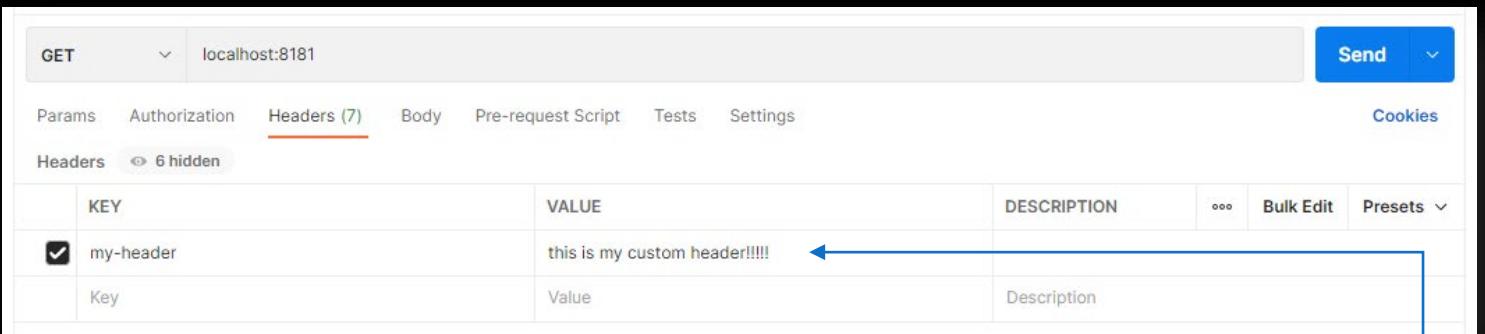
<http://expressjs.com/en/5x/api.html#req>



```

89  app.use((req, res, next) => {
90    console.log(chalk.yellowBright("request headers: "), req.headers);
91    next();
92 });

```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

[nodemon] starting `node .`
Listening on: http://localhost:8181
request headers: {
  'my-header': 'this is my custom header!!!!',
  'user-agent': 'PostmanRuntime/7.29.2',
  accept: '*/*',
  'postman-token': '3a356c3d-ed57-44bf-b252-bc1f77fe1040',
  host: 'localhost:8181',
  'accept-encoding': 'gzip, deflate, br',
  connection: 'keep-alive'
}

```

request.headers

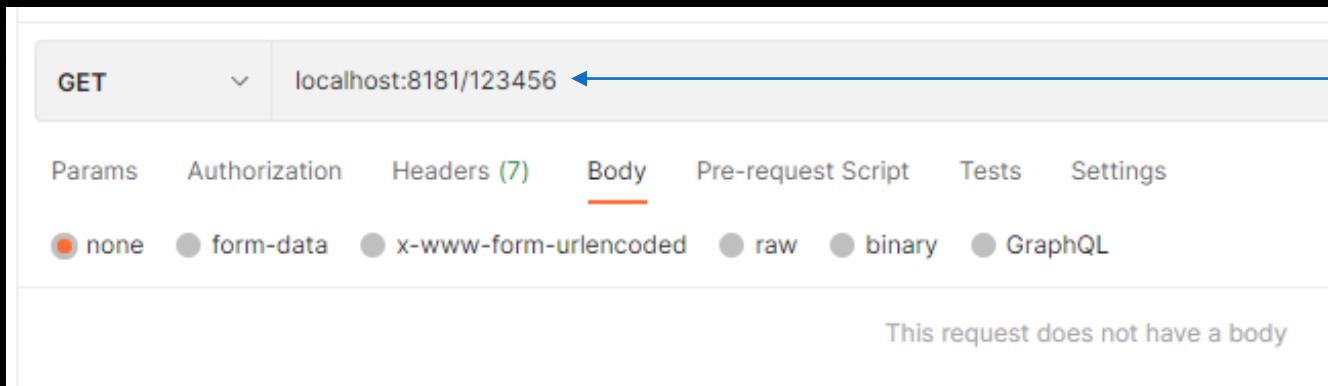
call back מעביר לפונקציות Express
שלו את אובייקט ה - `request` שאחראי
על הטיפול במידע שמאגיע בבקשת ה -
`http` שירותה.
דוגמה של להלן:

- נשתמש באובייקט ה - `request` כדי
להדפיס בקונסול את מפתח ה - `headers`
שנשלח בבקשת ה - `http`
- בתוכנת postman נרשום header חדש
משלנו ניתן לו שם (`my-header`) וערך
(`this is my custom header!!!!`)
- כשנלחץ על כפתור send יודפס לנו
בקונסול אובייקט ה - `headers` ובתוכו גם
ה - `header` שיצרנו.

ישנם `headers` שנכתבים על ידי express וישלחו באוטומטי
באובייקט התשובה.

req.params

```
94 app.use("/:id", (req, res, next) => {  
95   console.log(chalk.yellowBright("request params: "), req.params);  
96   next();  
97 });
```



```
Listening on: http://localhost:8181  
request params: { id: '123456' }
```

מטרה שנותנת לנו גישה לאובייקט `params` בשורת הכתובות.

דוגמה של להלן:

- בפרמטר הראשון אזכיר شبשורת הכתובות אני מצפה לקבל מפתח באובייקט ה - `params` בשם `id` בפונקציית ה - `call back` אדפיס את אובייקט ה - `params` בתוכנת `postman` אני מוסיף לשורת הכתובת לאחר הסימן / מספר שייכנס לתוך אובייקט ה - `params` כשליחן על כפטור `send` יודפס לנו בקורסול אובייקט ה - `params` ובתוכו גם המפתח `id` שיצרנו והערך מתוך שורת הכתובות שכתבנו.

req.query

```
104 app.use((req, res, next) => {  
105   console.log(chalk.yellowBright("request query params: "), req.query);  
106   next();  
107});
```

The screenshot shows the Postman interface with a GET request to `localhost:8181?user_id=123456&user_age=27`. The 'Params' tab is selected, showing the 'Query Params' section with two entries:

KEY	VALUE
<input checked="" type="checkbox"/> user_id	123456
<input checked="" type="checkbox"/> user_age	27
Key	Value

```
Listening on: http://localhost:8181  
request query params: { user_id: '123456', user_age: '27' }
```

מטרה שנותנת לנו גישה לאובייקט `query params` בשורת הכתובות.

בדוגמה של להלן:

- בפונקציה ה - `call back` אדרפיס את אובייקט ה - `query params` –
- בתוכנת Postman אני מוסיף לשורת הכתובות:
 - הסימן ? מסמן את תחילת ה - `query`
 - הסימן = מסמן את הערך של המפתח שמוופיע מצד שמאל של הסימן
 - הסימן & תחילת ה - `query` הבא
- כשלחץ על כפתור `send` יודפס לנו בקונסול אובייקט ה - `query params` – ובתוכו המפתחות שיצרנו.

Custom key in req

```
109 app.use((req, res, next) => {  
110   req.user = { name: "user", _id: 123456 }; ←  
111   console.log("request new key to request object named user: ", req.user);  
112   next();  
113 });
```

```
[nodemon] starting `node .`  
Listening on: http://localhost:8181  
request new key to request object named user: { name: 'user', _id: 123456 }
```

האובייקט של `request` הוא אובייקט javascript לכל דבר ועניין. על כן אני יכול לעשות השמה של מפתחות נוספים כראות עיני

בדוגמה של להלן:

- בפונקציית ה - `call back` נדרש מפתח חדש בתוכו אובייקט ה - `req` בשם `user` וואשווה אותו לאובייקט עם המפתחות והערכים הנ"ל
- נapis בקונסול מחרוזת תווים הכוללת את האובייקט שיצרנו
- כSAMPLE על כפתור `send` בתוכנת postman יודפס לנו בקונסול מחרוזת התווים עם האובייקט שיצרנו.

request.body

The screenshot shows two parts: a code editor and a Postman request interface.

Code Editor:

```
55 app.use((req, res, next) => {  
56   console.log("request body:", chalk.yellow(req.body)); ←  
57   next();  
58 });
```

Postman Request:

- Method: GET
- URL: localhost:8181
- Body tab is selected.
- Content type: raw
- JSON content:

```
1 {  
2   "name": "david", ←  
3   "age": 47  
4 }
```
- Send button

Terminal Output:

```
Listening on: http://localhost:8181  
request body: undefined ←
```

המפתח body באובייקט ה – request –
אחראי על המידע שmagiu בגוף הבקשה

פעיל את מетодת `app.use` כדי לירוט
בקשות ונדפס בקונסול את המפתח
`req.body`

- בתוכנת postman נלחץ על כפתור הניוט
body בתוכו נלחץ על האפשרות של raw
ונבחר להעביר מידע מסוג json
- לאחר מכן נכתב את התוכן שברצוננו
להעביר מצד לקוח לצד שרת במקום
המיועד לכך
- אולם כשנלחץ על כפתור send לא נצלילה
ליירוט את תוכן הבקשה.
- הסיבה לכך היא בגלל שהוא חייבם
להגדיר קודם את סוג המידע שהשרת יכול
לקבל ב – body ורק אז להעביר לו את
המידע.

! כיצד קובעים לשרת את סוג המידע שיעבירו לו יוסבר בשקפים הבאים

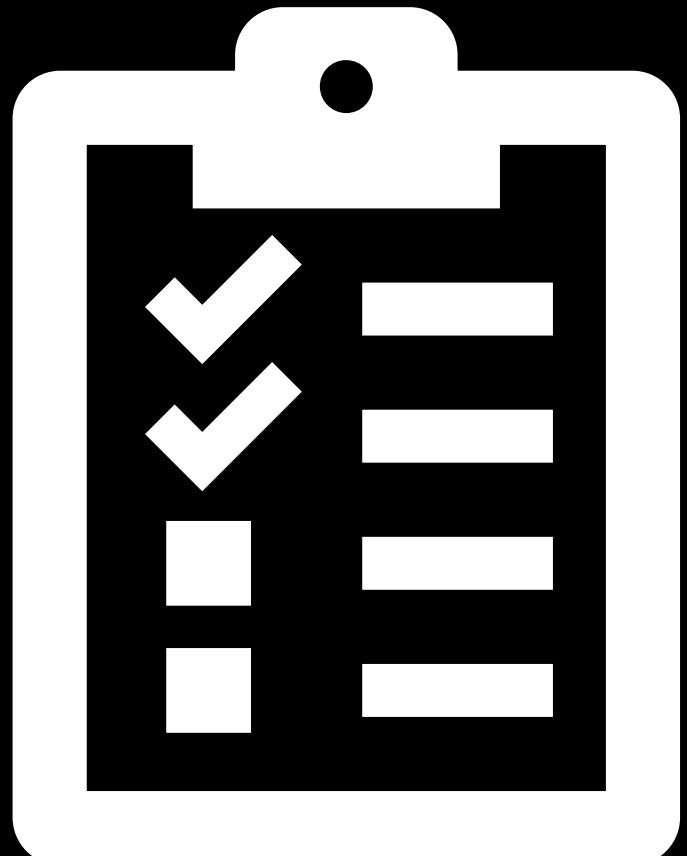
node-sandbox

- צור את המירטטים הבאים:

Method	url	response
Get	/headers	request headers
Get	/params/:id	request params
Get	/query	request query params
Get	/body	request body
Get	/custom	request custom user

- הפעיל את הקוד שבקובץ `entry point` באמצעות הפקודה המתאימה
- בדוק באמצעות תוכנת `postman` את שליחת כל בקשה זה – `http://localhost:3001` אליהן יוצרת מירטטים.

משימת request





Built-in middleware

פונקציות בינהים שנמצאות בתחום האובייקט של
express

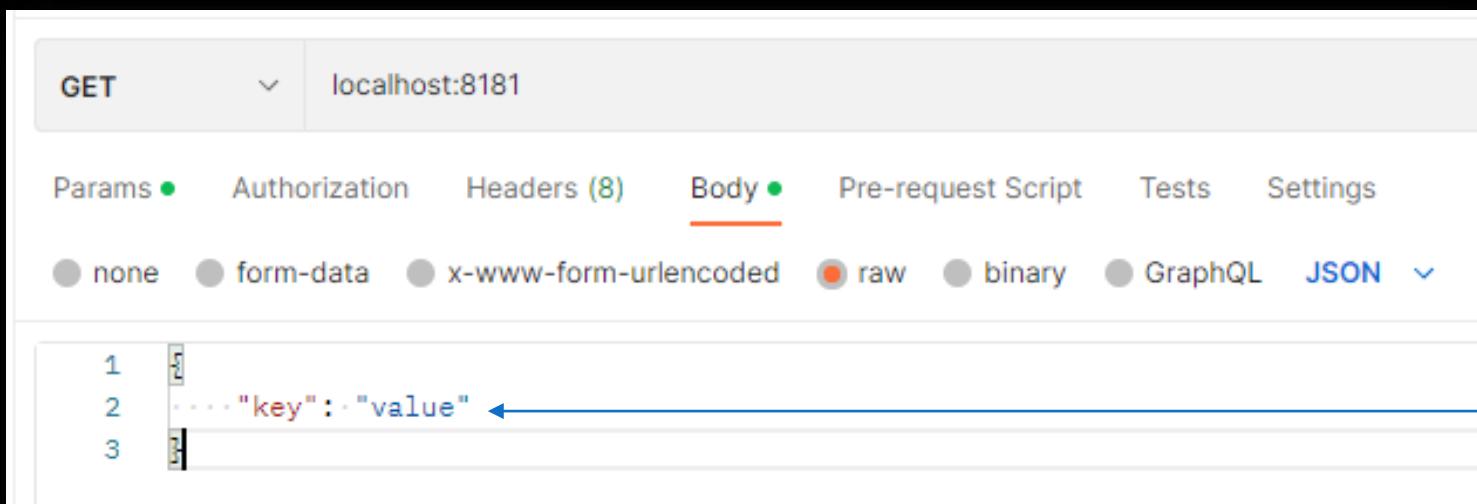
<http://expressjs.com/en/4x/api.html#express>



```

124 app.use(express.json()); ←
125
126 app.use((req, res, next) => { ←
127   console.log(chalk.yellowBright("request body:"), req.body);
128   next();
129 });
130
131 const PORT = 8181;
132 app.listen(PORT, () =>
133   console.log(chalk.blueBright("Listening on: http://localhost:8181"))
134 );

```



```

Listening on: http://localhost:8181
request body: { key: 'value' } ←

```

express.json

- מטודה שתעדכן את השירות של express שסוג מידע הוא מוכן לקבל הוא אובייקט של `JSON`:
- נפעיל את מטודת `express.json` בתוכה - middleware `app.use` ונסים אותו לפני היורティם האחרים של הכתובות על מנת שהשרת ידע לקבל את תוכן הבקשות HTTP מהסוג של `JSON`
- נפעיל מיירט שידפים לנו את גוף הבקשה נשלח אובייקט `JSON` תקני בגוף הבקשה ניתן לראות בקונסול כי הפעם השירות קיבל את האובייקט שהעבכנו לו בפתחה - `body`

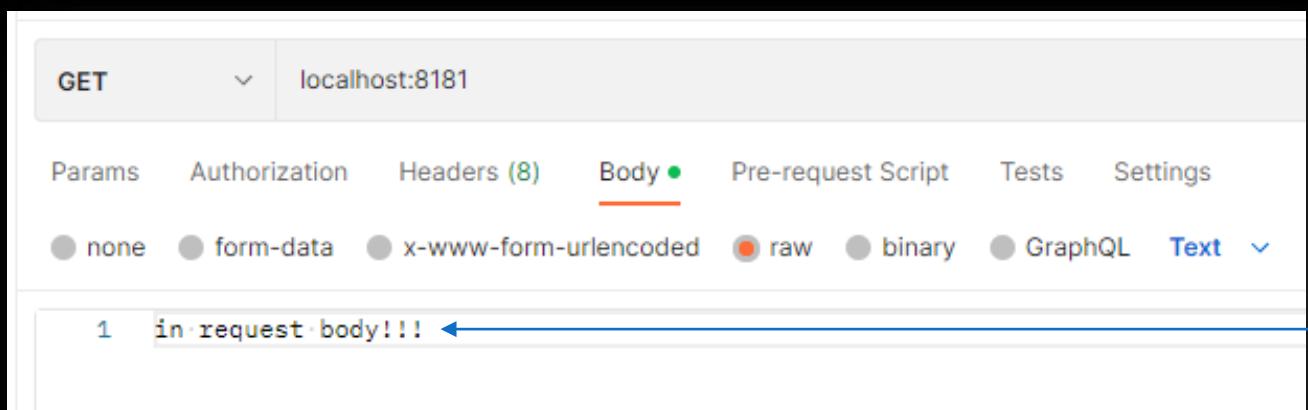
express.text

על מנת שהשרת יוכל לקבל גם הودעות מסווג של טקסט נפוץ מטודה נוספת מתוך אובייקט express והיא מטודה .text.

- נפעיל את מטודה express.text בתוך ה - middleware של app.use ונשים אותו לפני היירוטים האחרים של הכתובות על מנת שהשרת ידע לקבל את תוכן הבקשות HTTP מסווג של מחויבות תווים
- נשתמש באותו מיריט שידפים לנו את גוף הבקשה
- נשלח מחויבות תווים בגוף הבקשה
- ניתן לראות בקונסול כי הפעם השרת קיבל את מחויבות התווים שהעברנו לו בפתח ה body -

ישנה מטודה נוספת שמאפשרת לקבל מידע מיידע מסווג אחר בשם express.urlencoded הסביר על המטודה ב קישור הבא:
<http://expressjs.com/en/5x/api.html#express.urlencoded>

```
124 app.use(express.json());
125 app.use(express.text()); ←
126
127 app.use((req, res, next) => {
128   console.log(chalk.yellowBright("request body:"), req.body); ←
129   next();
130 });
131
132 const PORT = 8181;
133 app.listen(PORT, () =>
134   console.log(chalk.blueBright("Listening on: http://localhost:8181"))
135 );
```



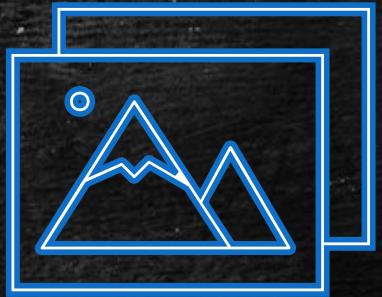
```
[nodemon] starting `node .
Listening on: http://localhost:8181
request body: in request body!!! ←
```

Business-cards-app

- אפשר העברת json לשרת באמצעות המethod המתאימה
- באמצעות תוכנת postman שלח בקשה HTTP לשרת עם מטודת post ועט אובייקט ה – json עם המפתחות שנמצאים בטבלה מצד ימין
- החזר לגולש תשובה עם האובייקט ששלחת

משימת express.json

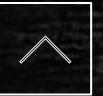
No.	key	Value type
1.	title	String
2.	subtitle	String
3.	description	String
4.	phone	String
5.	email	String
6.	Web	String
7.	Image	object
7.1	url	
7.2	alt	
8.	address	Object
8.1	state	String
8.2	country	String
8.3	city	String
8.4	street	String
8.5	houseNumber	Number
8.6	zip	Number



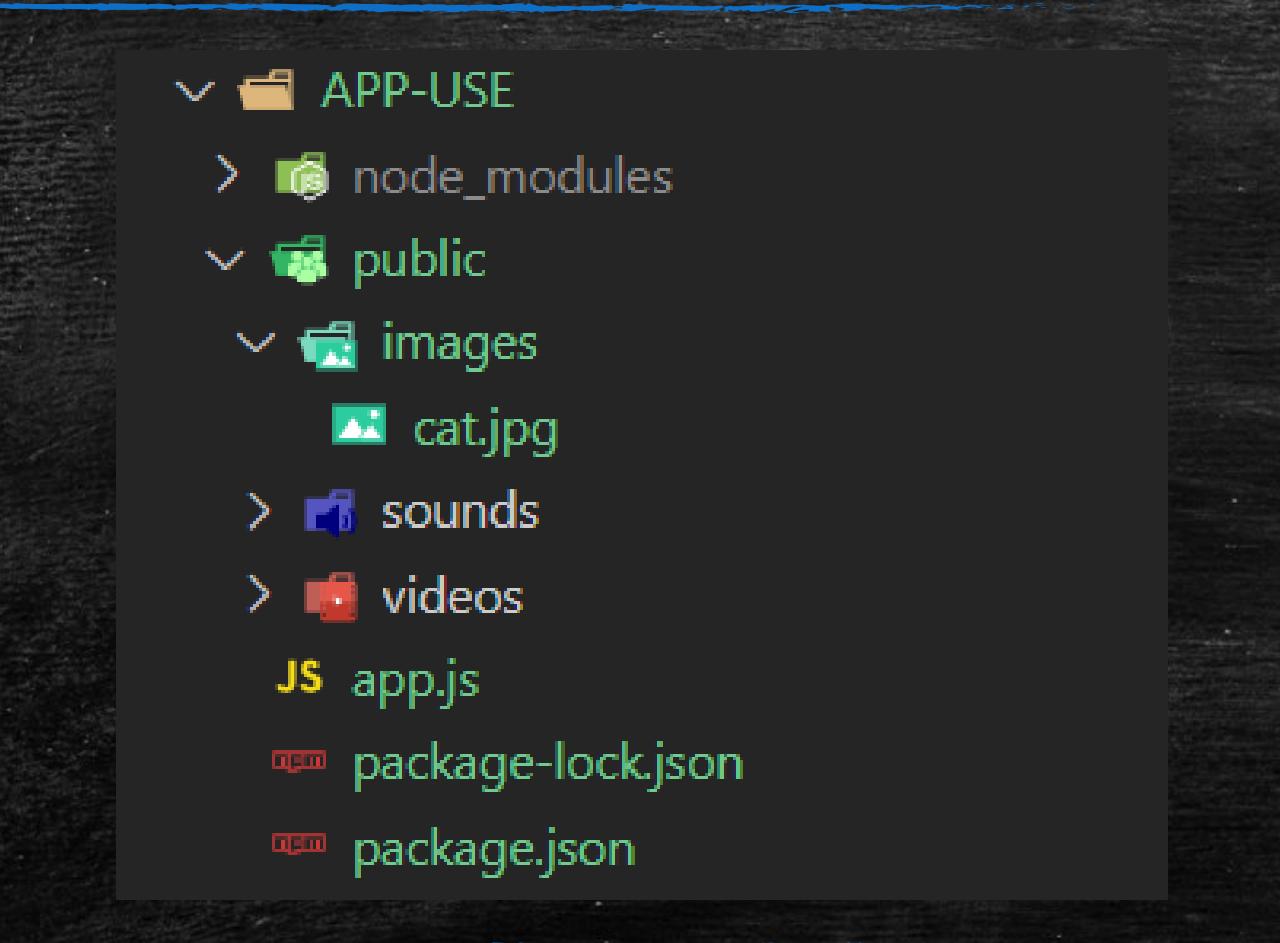
EXPRESS STATIC

מטודה המגישה קבצים סטטיים נשמרים בשרת.

<https://expressjs.com/en/5x/api.html#express.static>



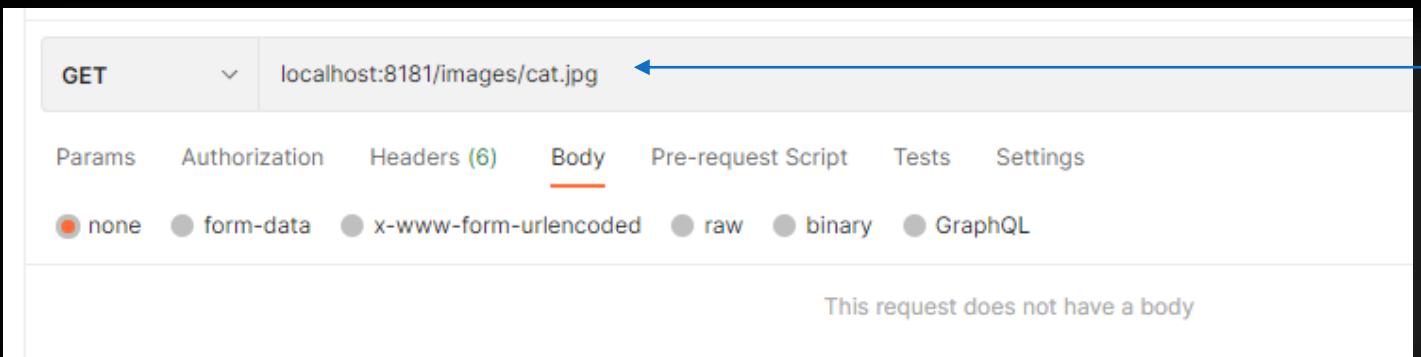
תשתיית קבצים



- בתוכן node.js-sandbox
- ניצוד את הנתיב `/public`
- ניצוד בתוכן נתיב זה את התיקיות:
 - images
 - sounds
 - videos
- יכניס בתוכן הנתיב `/public/images` קובץ של תמונה

express.static

```
143 app.use(express.static("./public")); | ←  
144  
145 const PORT = 8181;  
146 app.listen(PORT, () =>  
147   console.log(chalk.blueBright("Listening on: http://localhost:8181"))  
148 );
```



- מетодה `express.static` מקבלת בפרמטר הראשון את מיקום התיקייה שבה יוכנסו קבצים סטטיסיים כמו תמונות, סאונד וידאו pdf וכדומה ובפרמטר השני (אופציונלי) אובייקט קונפיגורציות

- כשאני אשלח בקשה לכתובת של השרת עם המיקום רלוונטי של התמונה (לא צוין תיקיית `public`) אם הכתובת לא תיורט על ידי מירט אחר `express` תחפש את הכתובת בתוך תיקיית `public` ואם היא תמצא היא תחזיר את התמונה המבוקשת

התוצאה

Body Cookies Headers (10) Test Results

Status: 200 OK Time: 30 ms Size: 424.8 KB Save Response ▾



🕒 Online ⌂ Find and Replace 📄 Console

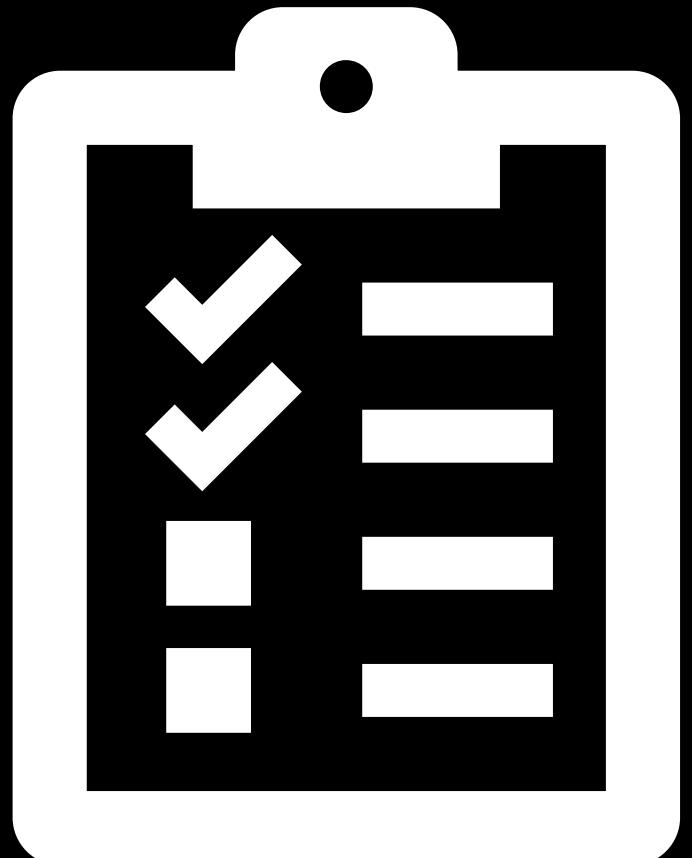
🍪 Cookies ⏷ Capture requests 📁 Bootcamp 🏁 Runner 🗑 Trash

Express
שנהברתי לה מחזירה אותה בתגובה
כך שניתן להציגה לגולש

Business-cards-app

- צור בפרויקט תיקייה בשם public
- צור בתחום שלוש תיקיות של:
 - pixabay - עם תמונה jpg מאתר images
 - pixabay - עם קובץ אודיו מסוג mp3 מאתר sounds
 - pixabay - עם קובץ וידאו מסוג mp4 מאתר videos
- express.static – הפעיל את מетодת server.js
- שלח שלוש בקשות בעזרת תוכנת postman כאשר:
 - הראונה תראה את התמונה
 - השנייה תשמש את קובץ הסאונד
 - והשלישית תראה את הסרטון

שימוש express.static





Error-handling middleware

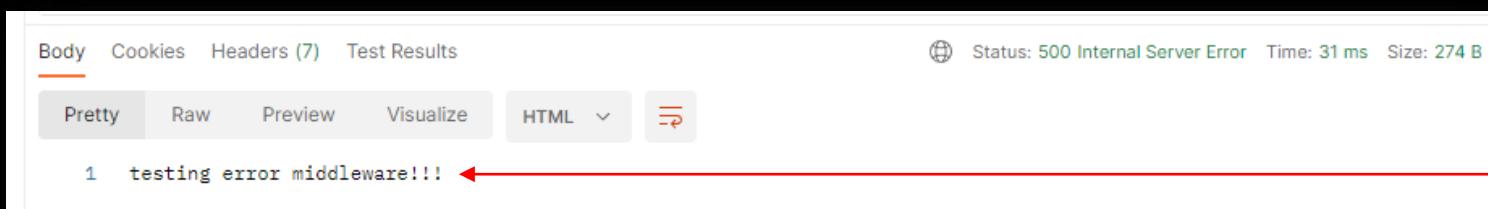
<http://expressjs.com/en/guide/using-middleware.html#middleware.error-handling>

פונקציות בינוים לטיפול בשגיאות שמקורו
בشرط באמצעות express



Handling server errors

```
132 app.use((req, res, next) => { ←  
133   | throw new Error("testing error middleware!!!");  
134 });  
135  
136 app.use((err, req, res, next) => { ←  
137   | console.error(chalk.redBright(err.message));  
138   | res.status(500).send(err.message);  
139 });  
140  
141 const PORT = 8181;  
142 app.listen(PORT, () =>  
143   | console.log(chalk.blueBright("Listening on: http://localhost:8181"))  
144 );
```



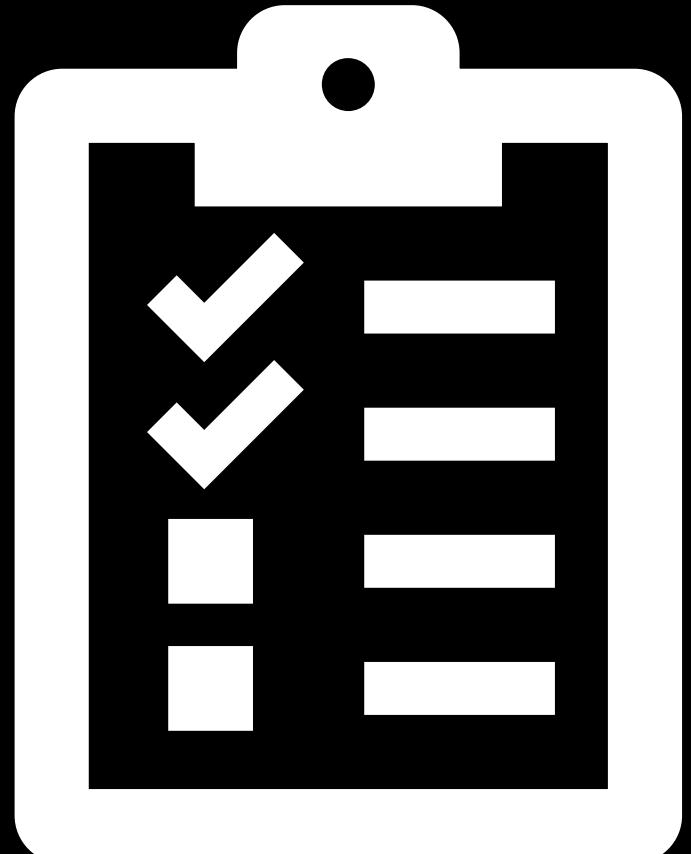
```
[nodemon] starting `node .`  
Listening on: http://localhost:8181  
testing error middleware!!!
```

- נפעיל את מетодת `app.use` ובתוכה
- נזרוק שגיאה
- על ידי העברת פרמטר דביעי
- לפונקציית ה - `call back` אני יכול
- לירות שגיאות שמקורן בשרת:
- המירט שיזופיע לנו את הודעת השגיאה
- מתוך אובייקט השגיאה שנפתח ויחזיר
- סטאטוס קוד 500 עם הודעת השגיאה
- נשלח את הבקשה
- כפי שניתן לראות בדוח הנטורנום יורה ונזרקה
- שגיאה שנפתחה בפונקציה שתפקידה
- לתחושים אותה, חזרה לגולש הודעת
- השגיאה שגם הודפסה בטרמינל בצדען
- אדום

Business-cards-app

- צור הנටיב `/utils/errorHandler.js`
- צור בתוכה התיקייה קובץ בשם `errorHandler.js` ובתוכו:
 - ייבא את ספריית `chalk`
 - צור את המטודה `handleError`
 - המטודה מקבל שלושה פרמטרים:
 - `response` - אובייקט ה-
 - `status` - סטאטוס הקוד של השגיאה
 - `message` - הודעה השגיאה עם ערך דיפולטיבי של מחרוזת תווים ריקה
 - המטודה תבצע בקונסול בצבע אדום בהיד את הודעה השגיאה
 - תחזר סטאטוס לפי הפרמטר סטאטוס שהועבר לה
 - ותחזר תשובה לגולש סטאטוס קוד והודעת השגיאה שקיבלה בפרמטר
 - ייצא את הפונקציה שיצרת מהמודול `server.js` - הוסף מירט לשגיאות שמקורן מהשרת והשתמש בפונקציה `handleError` עם אובייקט התשובה, סטאטוס 500 ועם הודעה השגיאה שמתקבלת בשגיאה

משימה server errors





Router-level middleware

פונקציות בינויים של express שמתחסkat
בניתובים באמצעות הפונקציה router

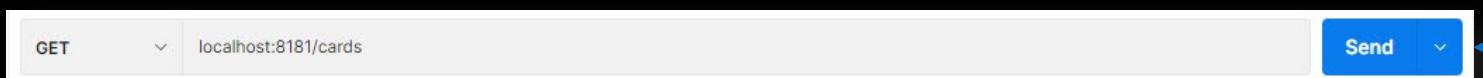
<http://expressjs.com/en/guide/using-middleware.html#middleware.router>



express.Router

JS app.js X

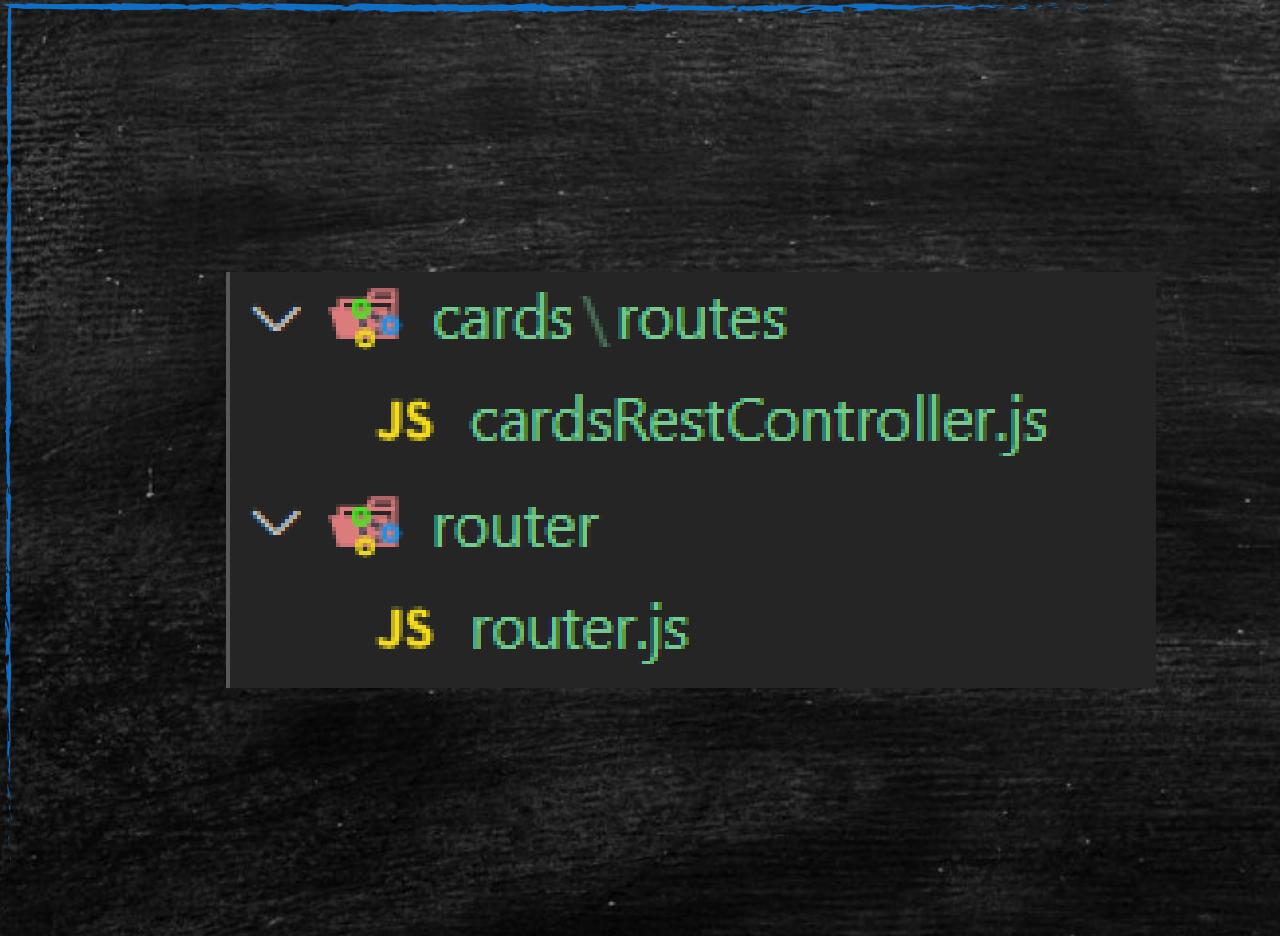
```
EXPRESS-ROUTER > JS app.js > ...
1  const express = require("express");
2  const app = express();
3  const router = express.Router(); ←
4  const chalk = require("chalk");
5
6  router.get("/", (req, res, next) => { ←
7    console.log("in cards!");
8    next();
9  });
10
11 app.use("/cards", router); ←
12
13 app.use((err, req, res, next) => {
14   console.error(chalk.redBright(err.message));
15   res.status(500).send(err.message);
16 });
17
18 const PORT = 8181;
19 app.listen(PORT, () =>
20   console.log(chalk.blueBright("Listening on: http://localhost:8181")));
21 );
```



```
[nodemon] starting `node .
Listening on: http://localhost:8181
in cards!
```

- ניצור מופע מהמחלקה `express.Router` וನשמר את הערך שיחזור בתחום המשתנה `router`
- נוסיף לאובייקט `router` את METHOD `get` שתДЕיס את מחוROTת התווים ותפניל את METHOD `next`
- משתמש ב – middleware של `app.use` ונדיר שם שורות הכתובת תתחילה במחוROTת התווים שבפורמט הראשון תופעל הפונקציה של `router`
- בתוכנת postman נשלח בקשה get לכתובת הרשומה
- ניתן לראות בקונסול של `app.use` יירטה את הבקשת, חתכה את שורת הכתובת שבפורמט הראשון והפנילה את METHOD `get` שחייבה את המטוROTת התווים בקונסול וכשמצאה אותה הדפסה את מחוROTת התווים בקונסול

תשתיית קבצים



Business cards app

- ניצoor את הנתיב `/cards/routes`
 - בתוכו ניצoor את הקובץ `cardsRestController.js`
- ניצoor את הנתיב `/router`
 - ובתוכו את הקובץ `router.js`

cardRestController.js

JS cardsRestController.js M X

cards > routes > JS cardsRestController.js > ...

```
1 const express = require("express");
2 const router = express.Router();
3
4 router.get("/", (req, res) => {
5   res.send([{}]);
6 }
7
8 module.exports = router;
```

קובץ ה - cards end points של

- ניבא את מטודת express
- ניצור את הפונקציה שתחזור אלינו מהפעלת המחלקה express.Router ומשמוד את הערך שיחזר בתוך המשתנה router
- נוסיף את מטודת get לפונקציה router
- הפונקציה תקבל בפרמטר הראשון את הכתובת
- בפרמטר השני פונקציית call back שתחזיר מערך עם אובייקט
- ניצא את הפונקציה router

Router.js

קובץ שאחראי על ניתוב הבקשות ל -
end points

```
JS router.js M X
router > JS router.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const cardsRestController = require("../cards/routes/cardsRestController");
4
5  router.use("/cards", cardsRestController); ←
6
7  module.exports = router;
```

- ניבא את מטודת express
- ניצור מופע מהמחלקה express.Router ושמור את הערך שיחזור בתוך המשתנה router
- ניבא את מודול cardsRestController.js
- נשימוש ב – router.use של middleware שallow app.use רק שהיא עובדת שזהה למטודת GET ונדיר שאם שורט על אובייקט זה – router
- הכתובת תתחילה במחוזת התווים שבפרמטר הראשון "/cards" תופעל הפקציה של router שבקובץ cardsRestController

Server.js

- נירט את כל סוגי הבקשות לכל הכתובות וניתן לקובי ה - router לניהל אותו
- ניבא את המודול router לתוכן קבוע באותו שם
- פעיל את פונקציית הביניהם app.use כרשותל כל בקשה היא תפעיל את פונקציית ה router שיצרנו באותו שם call back –

```
JS server.js M X  
JS server.js > ...  
1 const express = require("express");  
2 const app = express();  
3 const chalk = require("chalk");  
4 const router = require("./router/router"); ←  
5 const { handleError } = require("./utils/errorHandler");  
6  
7 app.use(express.json());  
8 app.use(express.text());  
9 app.use(express.static("./public"));  
10 app.use(router); ←  
11  
12 router.use((error, req, res, next) => {  
13   handleError(res, 500, error.message);  
14 };  
15  
16 const PORT = 8181;  
17  
18 app.listen(PORT, () => {  
19   console.log(chalk.blueBright(`Listening on: http://localhost:${PORT}`));  
20 });
```

postman

The screenshot shows the Postman application interface. At the top, there is a header bar with the method (GET), URL (http://localhost:8181/cards), and a 'Send' button. Below the header, there are tabs for Params, Auth, Headers (6), Body, Pre-req., Tests, and Settings. The 'Body' tab is selected, indicated by an orange underline. Under the 'Body' tab, it says 'none'. In the main content area, there is a message 'This request does not have a body'. To the right, there is a status bar showing 200 OK, 24 ms, and 237 B. Below the status bar, there is a 'Save Response' button. The bottom section shows the response body in JSON format, with three items labeled 1, 2, and 3. An arrow points from the text 'נשלח את ה - end point שיצרנו' to the URL field in the header. Another arrow points from the text 'שציפינו לה.' to the JSON response body.

```
[{"id": 1, "name": "Card 1", "description": "A sample card"}, {"id": 2, "name": "Card 2", "description": "Another sample card"}, {"id": 3, "name": "Card 3", "description": "A third sample card"}]
```

נשלח את ה - end point שיצרנו

- נשלח את הבקשה לכתובת הנכונה לירוט
- כפי שניתן לראות קיבלנו את התשובה שציפינו לה.

Error 404

```
js router.js M X
router > js router.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const cardsRestController = require("../cards/routes/cardsRestController");
4  const { handleError } = require("../utils/errorHandler");
5
6  router.use("/cards", cardsRestController);
7
8  router.use((req, res) => handleError(res, 404, "Page not found!")); ←
9
10 module.exports = router;
```

בקובץ `route.js` ניצור מיירט לכל הכתובות שאינן להם `end point` בכל אפליקציה ונידע את הגולש

- בעדרת פונקציית הביניהם `app.use` מיירט את כל הבקשות לכתובות השונות בכל ה- `methods` שלא יורטו על ידי פונקציות הביניהם הקודמות ונחזיר בתשובה סטאטוס 404 עם מחרוזת התווים "Page not found!" בעזרת הפונקציה `handleError` שיצרנו!

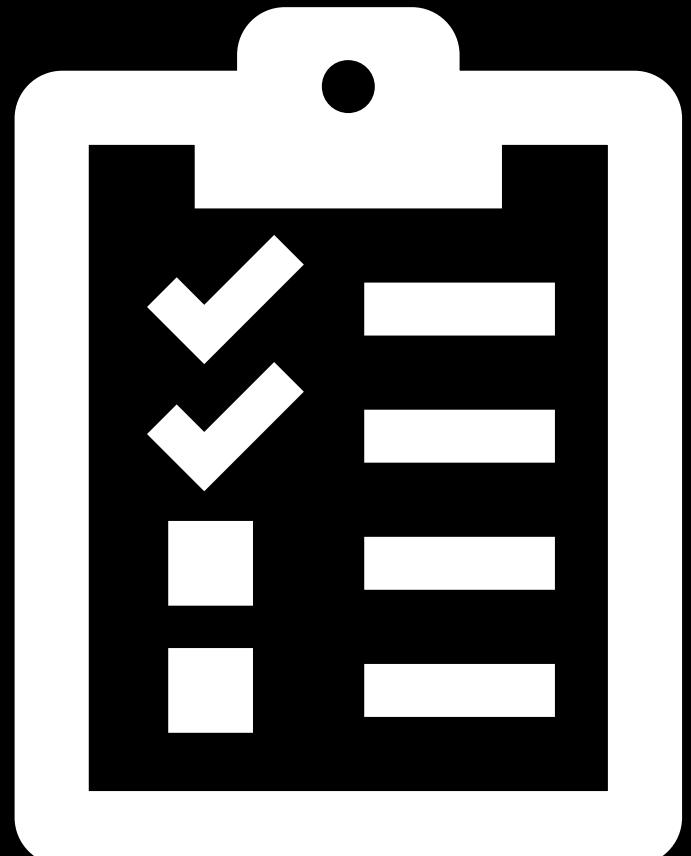
המקום של המיירט זהה חשוב וחיבר להיות לאחר כל המיירטים האחרים על מנת שלא ייחסם את הגישה למיירטים!

Business-cards-app

/cards/routes/cardsRestController.js

- בנתיב router.get שטיירט את הכתובת “/in cards get”
 - צור את המethodות הבאות:
 - router.get “/in cards get” מהריזת התווים נס “/” ותחזיר תשובה לדפס בקונסול את מחרוזת התווים נס “/” ותחזיר תשובה לגולש נס אותה מחרוזת תווים.
 - router.get “/in cards get params” מהריזת התווים נס “/” ותחזיר תשובה לגולש נס אותה מחרוזת תווים.
 - router.post “/in cards post” מהריזת התווים נס “/” ותחזיר תשובה לדפס בקונסול את מחרוזת התווים נס “/” ותחזיר תשובה לגולש נס אותה מחרוזת תווים.
 - router.put “/in cards put” מהריזת התווים נס “/” ותחזיר תשובה לדפס בקונסול את מחרוזת התווים נס “/” ותחזיר תשובה לגולש נס אותה מחרוזת תווים.

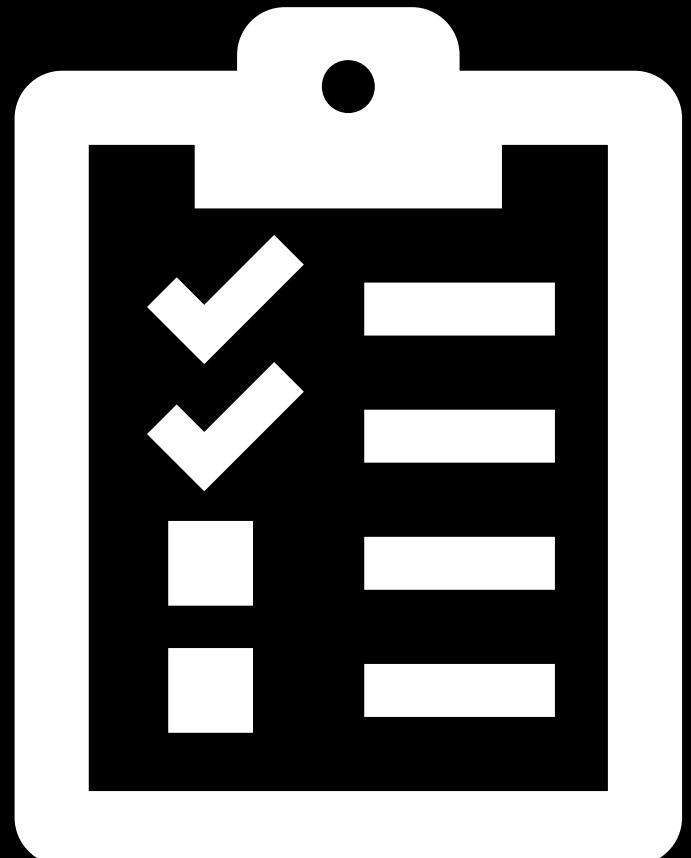
משימת Router



Business-cards-app

- id router.patch תיירט את הכתובת "/" עם param בשם id –
 - צור קבוע בשם pi והשווה את הערך שלו לערך pi מאובייקט ה- param.
 - הדפס בקונסול את מחרוזת התווים "in cards patch" ותחזיר תשובה לגולש עם אותה מחרוזת תווים.
- id router.delete תיירט את הכתובת "/" עם param בשם id –
 - הדפס בקונסול את מחרוזת התווים "in cards delete" ותחזיר תשובה לגולש עם אותה מחרוזת תווים.
 - יצא את מטודת router מהמודול

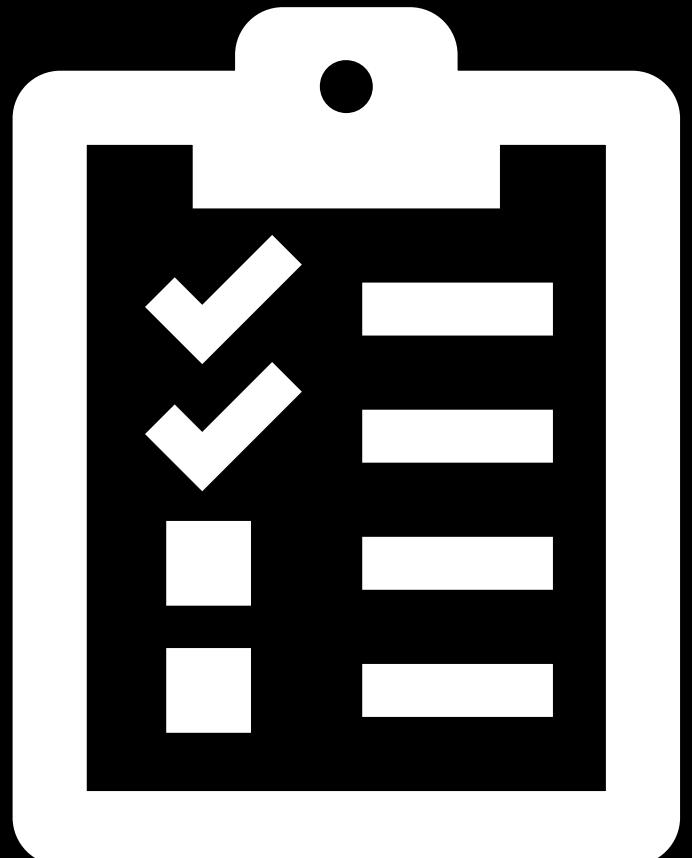
חלק ב'



Business-cards-app

- צור את הנטייב `/users/usersRestController.js` וצור את המטודות הבאות:
 - `router.post "/in users registration"` שטיירט את הכתובת “/” בקונסול את מחוזת התווים “`in users registration`” ותחזיר תשובה לגולש עם אותה מחוזת תווים.
 - `router.post "/login"` שטיירט את הכתובת “/login” בקונסול את מחוזת התווים “`in users login`” ותחזיד תשובה לגולש עם אותה מחוזת תווים.
 - ייצא את מטודת `router` מהמודול

חלק ג'



Business-cards-app

o בנתיב `/router/router.js`

- o ייבא את ה `router` שייצרת בקובץ `cardsRestController.js` למשתנה עם אותו שם
- o ייבא את ה `router` שייצרת בקובץ `usersRestController.js` למשתנה עם אותו שם
- o צור מידט בעזרת מетодת `router.use` לכתחובת `"/cards"` שיפעל את מетодת `router` שייבאת מהקובץ `cardsRestController.js`
- o צור מידט בעזרת מethodת `router.use` לכתחובת `"/users"` שיפעל את מethodת `router` שייבאת מהקובץ `usersRestController.js`
- o צור מידט שיידט את כל שאר הבקשות ויחזיר סטאטוס קוד 404 עם מחזורת התווים `"Page not found"` וудפס את מחזורת התווים בקונסול
- o יצא את מethodת `router` מהמודול `server.js`
- o ייבא את מethodת `router` לתוך משתנה בשם `router`
- o צור מידט לכל הכתובות בכל המethodות שיפעל את פונקציית `router` מתוך הקובץ `router`

חלק ד'





Third-party middleware

פונקציות מצד שלישי שיודעות לקבל את האובייקט של express ולהגיב לתוכנו

<http://expressjs.com/en/resources/middleware.html>



CORS

Cross-Origin Resource Sharing

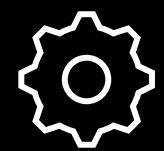
<http://expressjs.com/en/resources/middleware/cors.html>





Definition

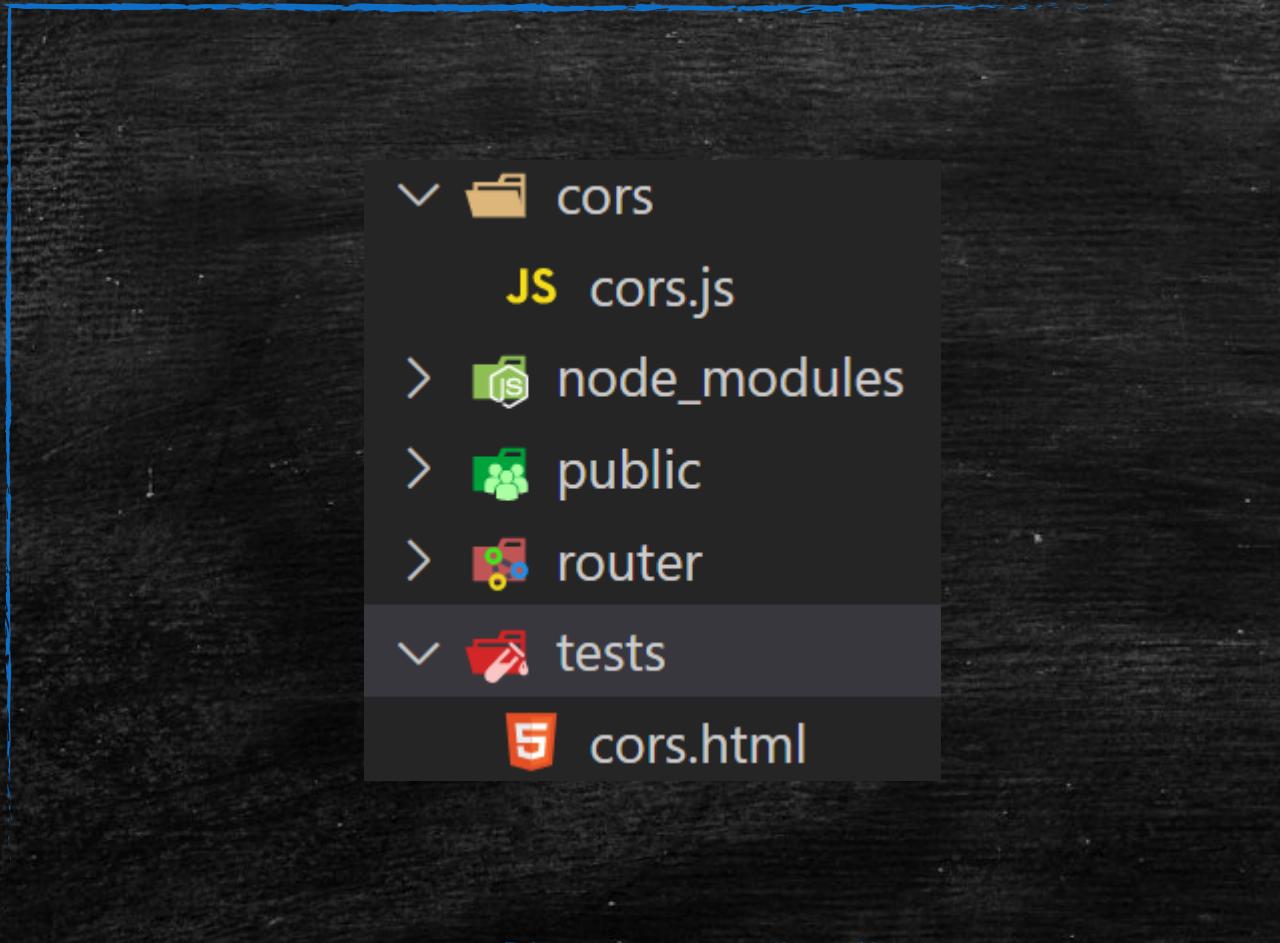
CORS is a part of HTTP that lets servers specify any other hosts from which a browser should permit loading of content



installation

npm i cors

תשתיית קבצים



Business cards app

- ניצור את הנתיב cors/cors.js
- ניצור את הנתיב /test/cors
- בתוכו ניצור את הקובץ cors.html

CORS > 5 cors.html > ...

```

1   <!DOCTYPE html>
2   <html lang="en">
3   <head>...
4   </head>
5   <body>
6     <pre id="output"></pre> ←
7
8     <script>
9       const output = document.getElementById("output");
10
11      async function getData() {
12        try {
13          const data = await fetch("http://127.0.0.1:3001");
14          const { message } = await data.json(); ←
15          return resultMessages(message); ←
16        } catch (err) {
17          resultMessages(err.message); ←
18        }
19      }
20
21      function resultMessages(message) {
22        console.log(message);
23        output.innerHTML = message;
24      }
25
26      getData(); ←
27    </script>
28  </body>
29</html>

```

cors.html

- הכנות התשתית:
- ניצור אלמנת עם id עם הערך output בו נעדכן את הגולש בהצלחה או כישלון הבקשה:
- פונקציה אסינכרונית השולחת בקשה fetch לכתובת ולפורט הבאים ושומרת אותו בתוך הקבוע data שיצרנו.
- נחלץ את מפתח message מתוך הערך שיחזור מהפעלת Method data.json()
- אם הגענו עד לשלב זה בקוד כלומר שלא קיבלנו שגיאה וכן נפעיל את Method resultMessages עם המשתנה message שהילצנו.
- אם תהייה שגיאה מנגן זה – try & catch שיצרנו יתפוצט את השגיאה ויפעל את Method resultMessages: resultMessages
- הפונקציה מקבלת הודעה
- מדפסה אותה לקונסול
- מעדכנת את הגולש בהודעה
- הפעלת פונקציית getData

JS cors.js X

```
CORS > JS cors.js > ...
1  const express = require("express");
2  const app = express();
3  const cors = require("cors");
4
5  app.use(
6    cors({
7      origin: "http://example.com",
8      optionsSuccessStatus: 200,
9    })
10 );
11
12 > // app.use( ...
13
14 app.get("/", (req, res, next) => {
15   res.json({ message: "success" });
16 }
17
18 app.listen(3001, function () {
19   console.log("Listening on: http://localhost:3001");
20 })
```

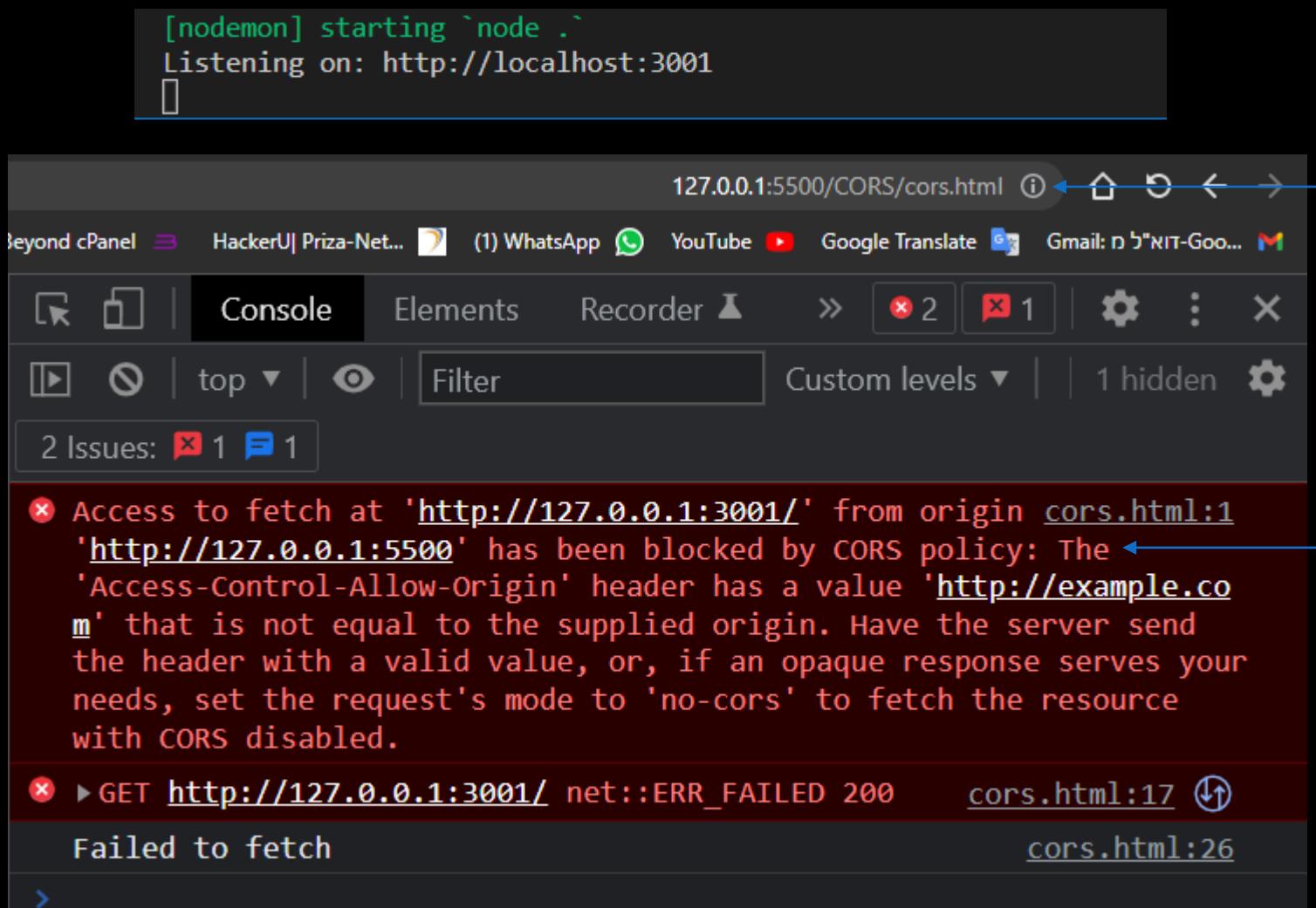
CORS.js

דוגמה לחסימת כתובות שאינה מוגדרת
באמצעות מטודת CORS

- לאחר שייבאתי את הספרייה באמצעות הפקודה `npm i cors` אני מייבא אותה למודול `:app.use`
- אני משתמש במטודת הביניהם – `app.use` כדי לירט כל בקשה שתשלח לשרת (חשוב להשתמש במטודה זאת בראש הדף)
- אני מנבייר בפורמט הראשון את הפעלת מטודת `cors` כאשר אני מנבייר לה אובייקט קונפיגורציות:
 - `origin` – הכתובת שמנחה תישלח הבקשה
 - `status` – איזה מספר ב – `optionsSuccessStatus` להחזיר אם תשלח בקשה מהכתובת שרשומה במפתח ה – `origin`

! במלחים אחרים בקשות יתקבלו רק מהכתובת הרשומה במפתח ה – `origin`

החוצה בקורס ונדפן



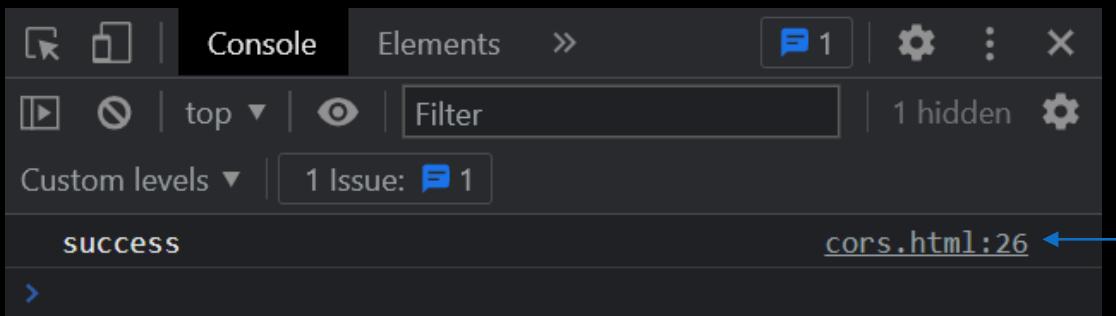
הmethod שמאפשר גישה לשרת המקומי
מוגדר מראש

- ניתן לראות לפי הודעת השגיאה כי הדפן חסם את הבקשה שנשלחה לשרת בכל שניסינו לשלוח אותה מהכתובת `http://127.0.0.1:5500/CORS/cors.h` ולא מהכתובת שהגדנו בmethod זה -
tml בשרת cors
(`http://example.com`)

JS cors.js M X

CORS > JS cors.js > ...

```
1 const express = require("express");
2 const app = express();
3 const cors = require("cors");
4
5 > // app.use( ...
11
12 app.use(
13   cors({
14     origin: "http://127.0.0.1:5500", ←
15     optionsSuccessStatus: 200,
16   })
17 );
18
19 > // app.use( ...
27
28 app.get("/", (req, res, next) => {
29   res.json({ message: "success" }); ←
30 });
31
32 app.listen(3001, function () {
33   console.log("Listening on: http://localhost:3001");
34 });
```



CORS.js

דוגמא לקבלת בקשות מכתובת מורשתית

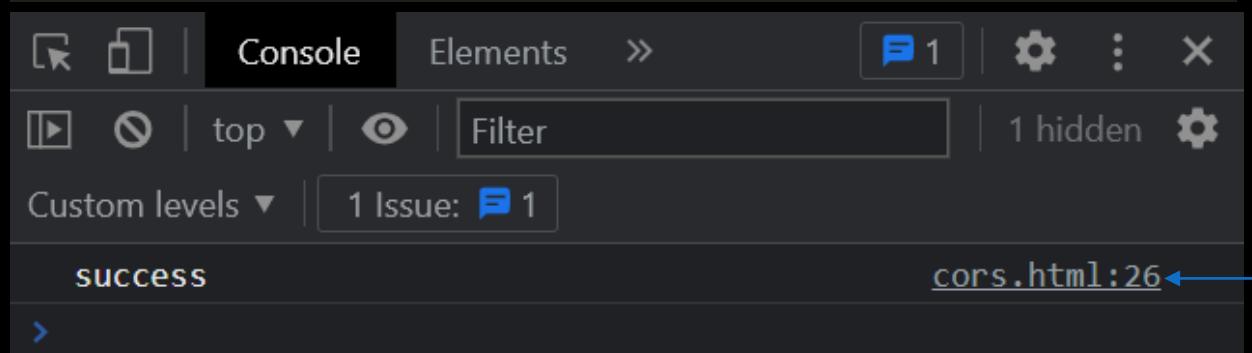
- הפונס נציגן באובייקט הקונפיגורציות של הפקצייה cors שאננואפשרים שליחת בקשות HTTP מהכתובת `http://127.0.0.1:5500`
- אני מירות את METHOD get (שדף ישלח) ושולח בתגובה אובייקט עם ההודעה `success` ניתן לראות שהבקשה יורתה בהצלחה מהכתובת שהגדרתי וקיבלה את התשובה המוצפה אותה פונקציית `resultMessages` הדפסה בקונסול

מערכת של כתובות

דוגמה להעברת מידע עם כל הכתובות
שאני מאפשר לקבל מהם בקשות HTTP

- מפתח ה – origin שבאובייקט cors הקונפיגורציות שאני מעביר למטרצת יכול לקבל או מחרוזת תווים או מידע של מחרוזות תווים של כתובות מאושרוות לשילוחת בקשות HTTP לשרת.

- ניתן לראות את יירוט הבקשה על ידי השרת והוורת התגובה המוצופה



```
JS cors.js M X
CORS > JS cors.js > ...
1 const express = require("express");
2 const app = express();
3 const cors = require("cors");
4
5 > // app.use(...
20
21 app.use(
22   cors({
23     origin: ["http://example.com", "http://127.0.0.1:5500"],
24     optionsSuccessStatus: 200,
25   })
26 );
27
28 app.get("/", (req, res, next) => {
29   res.json({ message: "success" });
30 });
31
32 app.listen(3001, function () {
33   console.log("Listening on: http://localhost:3001");
34 });

Console Elements » 1 | 1 hidden
Custom levels ▾ 1 Issue: 1
success
cors.html:26
```

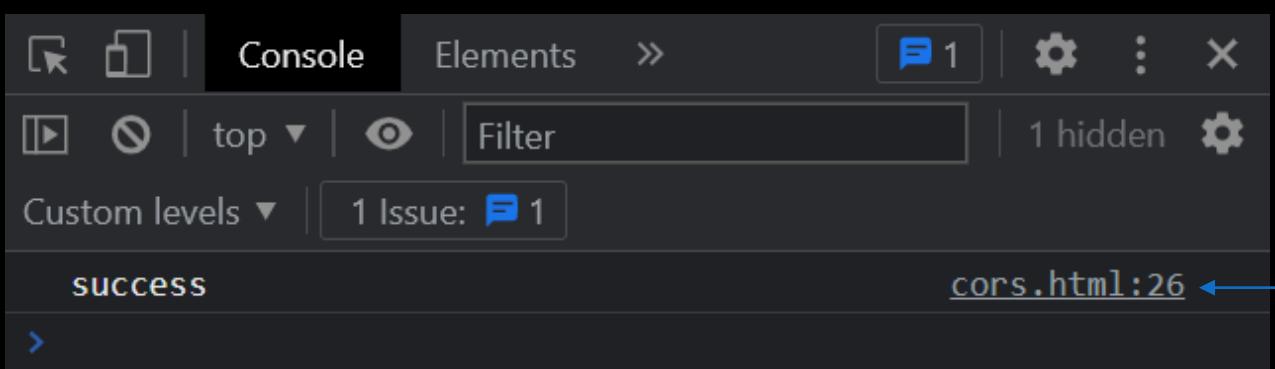
הרשה לכל הכתובות

דוגמה למתן הרשות לכל הכתובות לשולח בקשות HTTP לשרת

- אם אפניאל את מטודת cors ולא ענבר לה אובייקט קונפיגורציית היא לא אפשר שליחת בקשות לשרת מכל כתובות.

ניתן לראות שהשרת יירט את הבקשה למורות שלא הגדרנו בפונקציית ה – cors מאיזה כתובות נשלחת הבקשה

```
JS cors.js M ●  
CORS > JS cors.js > ...  
1  const express = require("express");  
2  const app = express();  
3  const cors = require("cors");  
4  
5  // app.use(...  
25  
26  app.use(cors()) ←  
27  
28  app.get("/", (req, res, next) => {  
29    | res.json({ message: "success" });  
30  });  
31  
32  app.listen(3001, function () {  
33    | console.log("Listening on: http://localhost:3001");  
34  });
```

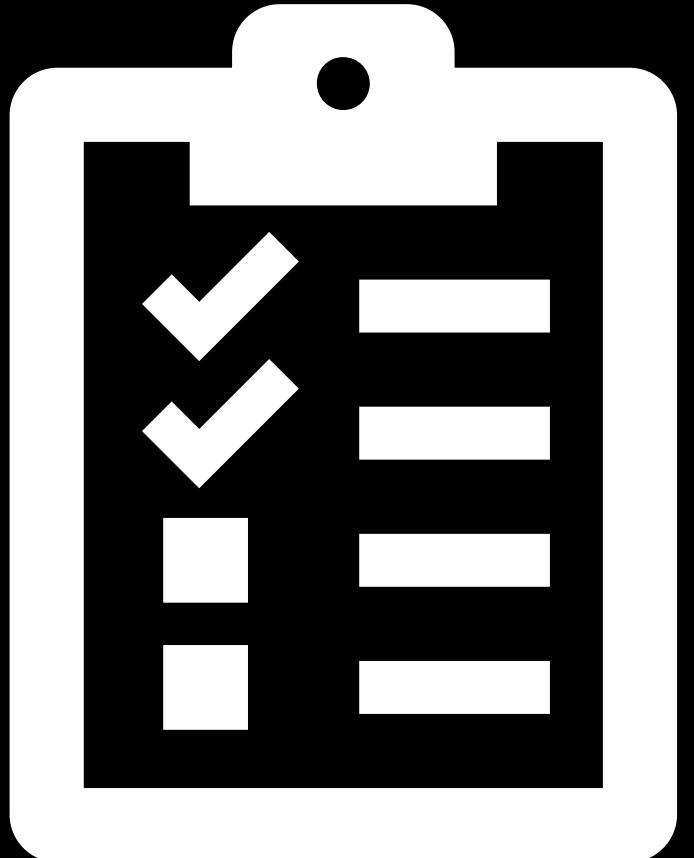


! לא נעלם את האפליקציה עם הפעלת מטודת CORS בצוותה הזאת בגל שהיא יוצרת פרצת אבטחה חמורה

Business-cards-app

- צור את הנתיב `/cors/cors.js`
- אפשר שליחת בקשה מכתובות הבאות
`http://127.0.0.1:5500` , `http://localhost:3000`
- server.js
 - ייבא את המודול של `cors.js`
 - צור מידת שיפעל את ה - `middleware` על כל בקשה שנשלחת לשרת

משימת CORS

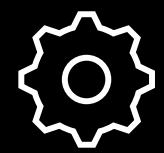


morgan

HTTP request logger middleware for node.js

<http://expressjs.com/en/resources/middleware/morgan.html>





installation

npm i morgan

morgan

הmethod המשמש כ-`logger` לבקשת
HTTP שנשלחת לשרת

- נשתמש בפונקציה בתוך `app.use(app)` על מנת
שתיירט כל בקשה בכל כתובות ובכל דרך
`(method)` וזאת כדי לקבל את שורת
הכתובת שממנה נשלחה הבקשה ונפעריל את
method `morgan` שמקבלת מחרוזת תווים
כאשר כל פריט מידע שנייתן להציג
בקונסול מופרד על ידי הסימן נקודתיים
התוצאה בקונסול עם יירוט הבקשה
מהדף

```
JS morgan.js M ×

JS morgan.js > ...
1 const express = require("express");
2 const app = express();
3 const chalk = require("chalk");
4 const cors = require("cors");
5 const morgan = require("morgan");
6
7 app.use(
8   morgan(`←
9     | chalk.cyanBright(":date[clf] :method :url :status :response-time ms")`←
10    | )←
11  );
12
13 app.use(
14   cors({
15     origin: "http://127.0.0.1:5500",
16     optionsSuccessStatus: 200,
17   })
18 );
19
20 app.get("/", (req, res, next) => {
21   res.status(200).json({ message: "success" });
22 });
23
24 app.use((err, req, res, next) => {
25   console.error(chalk.redBright(err.message));
26   res.status(500).send(err.message);
27 });
28
29 const PORT = 3002;
30 app.listen(PORT, function () {
31   console.log(chalk.blueBright("Listening on: http://localhost:3002"));
32 });

[nodemon] starting `node .` ←
Listening on: http://localhost:3002 ←
[30/Sep/2022:11:44:02 +0000] GET / 304 3.576 ms ←
```

! לרשימת האפשרויות במחוזות התווים:
<https://www.npmjs.com/package/morgan>

Morgan's call back

```
20 app.use(  
21   morgan((tokens, req, res) => { ←  
22     return [  
23       tokens.method(req, res),  
24       tokens.url(req, res),  
25       tokens.status(req, res),  
26       "- ",  
27       tokens["response-time"](req, res),  
28       "ms",  
29     ].join(" "));  
30   })  
31 );
```

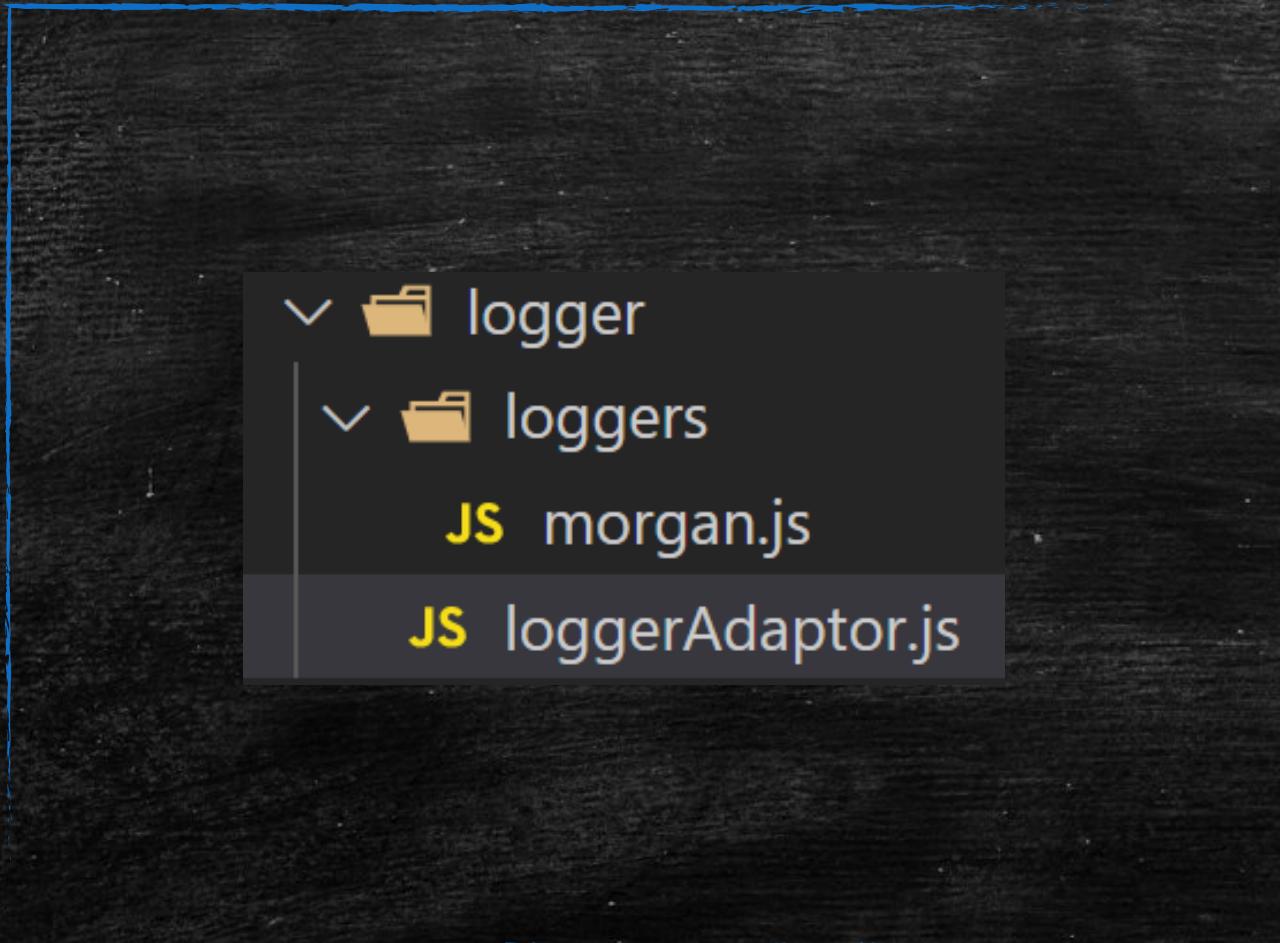
POST / 404 - 3.257 ms ←

פונקציה זאת יכולה לקבל או מחזרות
תווים כפי שראינו בשקף הקודם או
פונקציית call back

- Call back
- מקבלת שלוש פרמטרים:
 - Tokens – אובייקט שיכיל את המטודות
 - Req – אובייקט הבקשה
 - Res – אובייקט התגובה
- הפונקציהמחזירה מערך של מחזרות תווים
(כל מטודה של ה – tokens מחזירה מחזרות
תווים)
- הפונקציה morgan הופכת את מערך
מחזרות התווים למחזרות תווים ומדפיסה
אותן בקונסול.
- התוצאה בקונסול

היתרון בשיטה זאת הוא שהוא שnitן להפעיל לוגיקה בתוך
פונקציית ה – call back כך שאם תנאי מסוים מתקיים
תחזר מחזרות התווים בצורה מסוימת אחרית תחזר מחזרות
תווים בצורה אחרת !

תשתיית קבצים



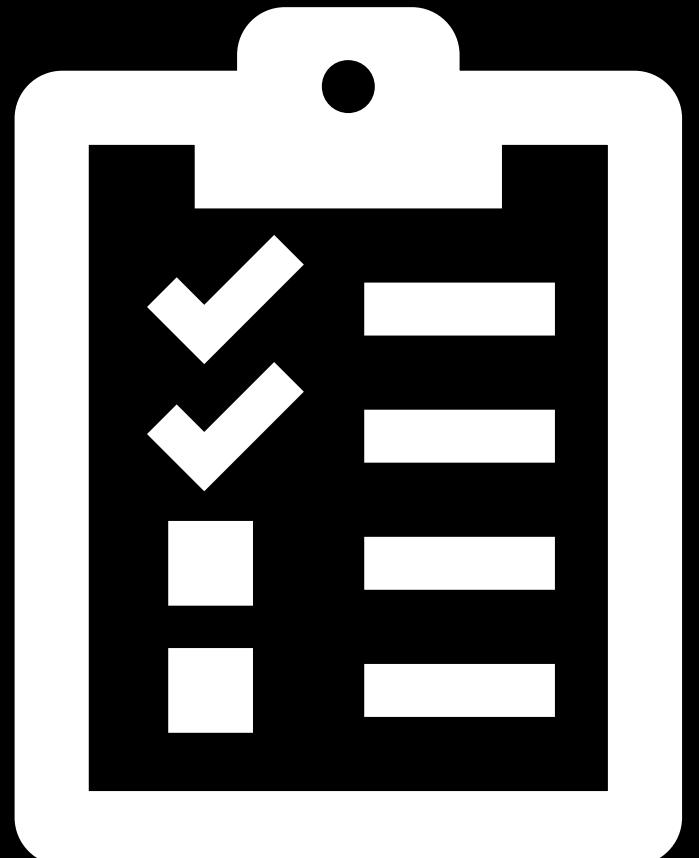
Business cards app

- ניצור את הנתיב
/logger/loggers/morgan.js
- ניצור את הנתיב
/logger/loggerAdaptor.js

Business-cards-app

- בנתיב `logger/loggers/morgan.js`
 - ייבא את הספריות `morgan chalk` למודול `morgan`:
 - צור קבוע בשם `logger` שערך יהיה שווה להפעלת מטודה:
 - המטודה מקבלת פרמטר `call back`
 - היא מקבלת שם ה – `status` התגובה הוא 400 ומעלה היא תחזיר את מערך מחווזות התווים הבא שצבוע בculo אדום בהיר:
 - תאריך ושעה של יירוט הבקשה
 - המטודה של הבקשה
 - לאיזה כתובות נשלחה הבקשה
 - הסטאטוס קוד של התשובה
 - כמה זמן ערכה התשובה
 - לאחר מכן היא תחזיר את אותו מערך תווים רק בצבע `cyanBright`
 - יצא את ה – `logger` שייצרת

משימת
morgan



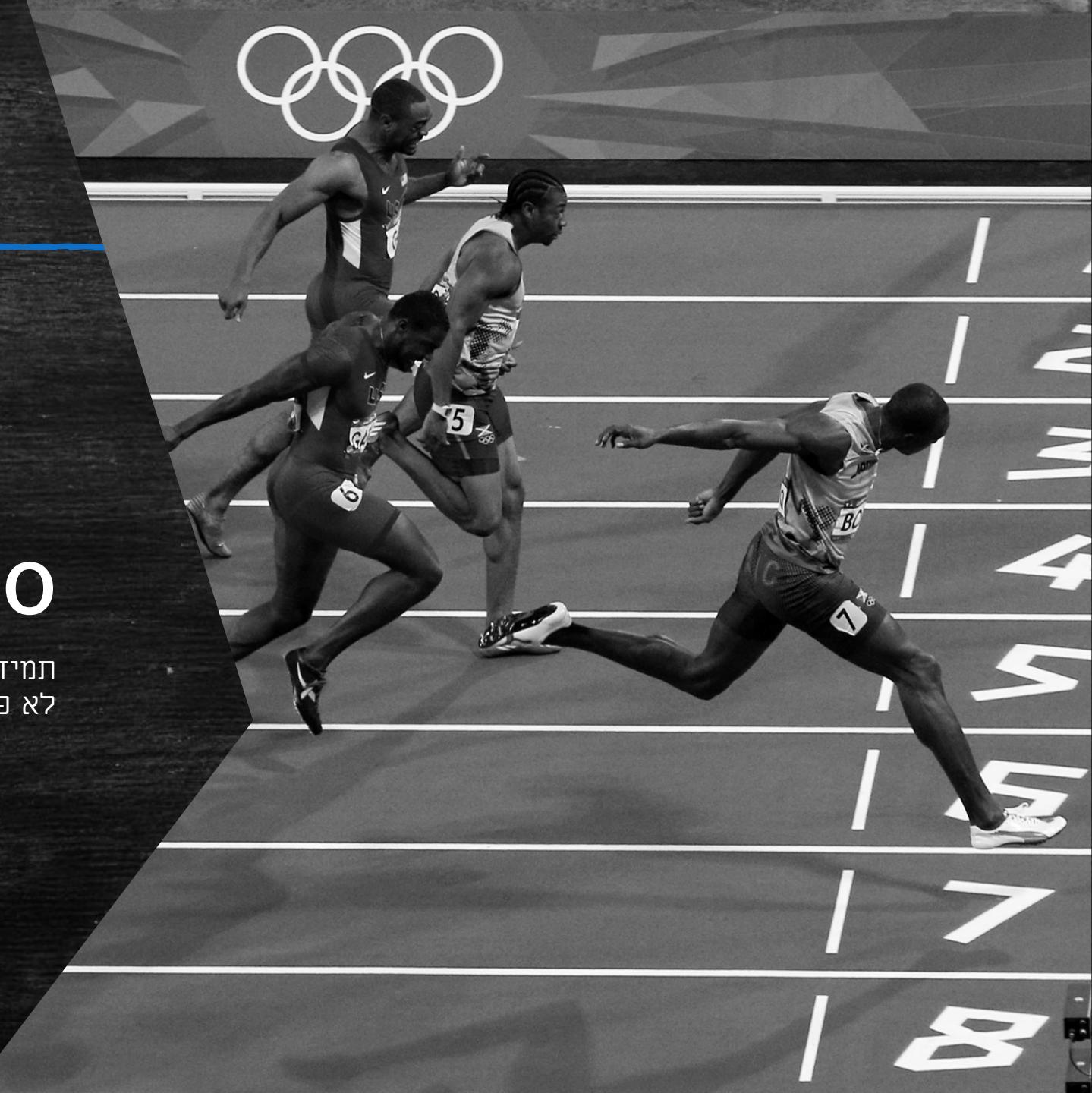
חלק ב'

- בנתיב `/logger/loggerAdaptor.js`
- ייבא ה – `logger` שייצרת מתוך הקובץ `morgan.js` אל תוך קבוע בשם `morganLogger`
- צור קבוע בשם `LOGGER` שיהיה שווה ל – `"morgan"`
- התחה שאם הערך של הקבוע `LOGGER` שווה ל – `"morgan"` תופעל פונקציית הבינאים `app.use()` ובתוכה הפונקציה `morganLogger`
- יצא את המטודה `app` מהמודול `server.js`
- ייבא את הפונקציה `loggerService` שיצאת מ – `logger` לתוך קבוע בשם `loggerService`
- הפעיל את פונקציית הבינאים `app.use()` שתירת כל בקשה בכל מטודה ותפעיל את פונקציית `loggerService`
- בדוק בעזרת התוכנה `postman` שליחת בקשה לשרת ואת הלוגר



וְיַעֲשֵׂה

תמיד אפשר להרחב לעדכן לשככל ולהוסיף עוד פיצ'רים אבל
לא פחות חשוב זה לדעת מתי לנצוץ.



שירה בהצלחה!