REST

REST

תוכן עניינים

1. מבוא ל - REST

REST (**RE**presentational **S**tate **T**ransfer) מהווה סטנדרט ברשת האינטרנט. הוא מבוסס על פרוטוקול HTTP ומטרתו לנסח את כללי העברת המידע והתקשורת ברשת, בין צד לקוח לצד שרת ואף בין שרתים עצמם. התקן הוצג לראשונה בשנת 2000 על ידי Roy Fielding אחד ממנסחי הפרוטוקול HTTP שמשמש לתיאור מסרים בין שני גורמים ברשת. מסרים לבקשת מידע ומסרים למילוי הבקשה מוכרים כ REQUEST RESPONE .

2. הגדרות ומושגים בסיסיים

רשת האינטרנט היא אוסף של מחשבים הקשורים ביניהם בתקשורת. ברשת ישנם שרתים המכילים מידע, וישנן תחנות עבודה אשר בעזרת דפדפנים, יכולים למשוך על ידי בקשה את המידע. המידע יכול להיות *סטאטי* (כגון תמונה, סרט, מאמר וכו') ו*דינאמי* (כגון רשימת קניות יומית אשר צריכה להיווצר על ידי אפליקציה הפועלת בשרת על ידי בקשה מתאימה). העברת המידע יכולה להיות בין שרת לתחנת עבודה (על ידי דפדפן המפרט מזהה המידע שנמצא בשרת) וגם בין שרת לשרת (למשל שרת אפליקציה יפנה לשרת נתונים לקבל נתונים אותם ישלח לתחנת עבודה לאחר עיבוד הנתונים). בבואנו לבחון את הארכיטקטורה בה אנו עובדים יש לפנינו את הגורמים הבאים:

- המשאבים והאפליקציות שאותם צריך לבקש
- כל משאב לזהות במדויק על ידי URI (Unique Resource Identifier)
- נוהל ההדברות בין שני מחשבים מנוסח על ידי פרוטוקול HTTP המוגדר על ידי גופי תקינה עולמים

2.1.1. המשאבים ואפליקציות הגדרה

כל תמונה, כל סרטון, כל דף מידע (HTML) (כגון דף הבית של האתר) מהווה משאב. משאבים אלו נקראים *סטאטיים*. לכל משאב (Resource) יש שם ב – file system כגון: ape.png , mytrip.mp3 , homepage.html וכדומה. בנוסף יש לשרתים אפשרות למספר משאבים דינאמיים, למשל פרטי לקוח עם רשימת ההזמנות שלו בחודש האחרון. ברור לנו שהלקוח הפעיל מזרים הזמנות מידי יום ולכן הנתונים ישתנו מבקשה לבקשה. מאחר ונתונים אלו חייבים להיווצר כל פעם מחדש הם נקראים דינאמיים. משאבים אלה מיוצרים על ידי אפליקציות בכלים שונים כגון NodeJS PHP וכדומה. כאשר מגיעה בקשה לשרת, היא גורמת לאפליקציה לעבוד וליצור את הנתונים על פי פרטי הבקשה, כלומר הפרמטרים שהועברו לבקשה. במידה ורוצים לקוח מסוים יש לספק את המזהה של הלקוח וכדומה.

2.1.2. התקן לזיהוי משאבים - URI URL (RFC-3986)

לכל משאב חייבים לתת שם הנקרא URI – UNIFORM RESOURCE IDENTIFIER. השם יכול לתאר משאב *פיזי*, כמו קובץ פיזי על דיסק, או משאב *לוגי*, כמו כל המכירות של היום. שם זה יקבל כינוי חדש URL - UNIFORM RESOURCE LOCATOR, כאשר הוא יכיל גם מידע כיצד ניגשים למשאב הנדון.
ה - URL מורכב מרכיבים מוסכמים וצריך לבנות ולנתח אותו לפי הכללים הללו.

להלן המרכיבים של המזהה לפי הדוגמא הבאה:

https://video.google.com:8080/videoplay?docid=1234&leng=en#00hh02m30s

https://	Protocol
video	Subdoamin
google.com	Domain name
8080	Port
videoplay	Path
?	Query
docid=1234&leng=en	Parameters
#00hh02m30s	Fragment

דוגמאות:

ftp://ftp.is.co.za/rfc/rfc1808.txt
בקשה לקבל קובץ טקסט בשם rfc1908.txt המאוחסן בשרת

http://www.ietf.org/rfc/rfc2396.txt

ldap://[2001:db8::7]/c=GB?objectClass?one
פניה למשאב בשרת עם ציון פרמטרים למיקוד הפניה

mailto:John.Doe@example.com
שליחת מייל לכתובת המפורטת

news:comp.infosystems.www.servers.unix
התחברות לשרת חדשות לשם קבלת נתונים*

tel:+1-816-555-1212

חיוג למספר המפורט

telnet://192.0.2.16:80/

כניסה עם תוכנת שירות telnet למחשב שכתובתו מפורטת עם ציון PORT

דוגמא לזיהוי משאב סטאטי:

http://www.my-site.co.il/homepge.html
http://www.flowers/inages/lili.jpg

דוגמא לזיהוי משאב דינאמי:

http://www.my-site.co.il/customer.aspx?custID=1234&orders=daily

בדוגמא זו מבקשים את פירוט ההזמנות של לקוח שמספרו 1234

http://www.my-site.co.il/customer.aspx?custID=1234

בדוגמא זו מבקשים את פרטי לקוח שמספרו 123

2.1.3. פרוטוקול השיחה HTTP תקן (RFC2616 , RFC7230 , RFC7231)

פרוטוקול HTTP (Hyper Text Transfer Protocol) שייך לשכבת היישום stateless. מבוסס על הודעות (messages).
יש שני סוגי הודעות:

1. הודעת בקשה הפונה לקבלת משאב נקראת request
2. ההודעה היא תגובה response שאותה שולח מקבל הבקשה, בדרך כלל שרת, בחזרה למבקש.

מבנה ההודעות הוא: יש כותרות בהן פירוט מאפיינים של הבקשה נקרא בשם head שורה ריקה ואחריה גוף ההודעה נקרא בשם body. גוף ההודעה מכיל את תוכן המשאב הנקרא payload.

דוגמא ל – Request:

A request: http://www.somehost.com/path/file.html

```
GET /path/file.html HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
[blank line here]
```

דוגמא ל – Response:

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354
[blank line here]
<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
</body>
</html>
```

HTTP – Methods/ verbs

HTTP Verb	**CRUD**	**Entire Collection (e.g. /customers)**	**Specific Item (e.g. /customers/{id})**
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

2.1.4. מה התחדש ב - REST

ראינו שקיימים משאבים לרוב במרחבי הרשת, וזיהוי משאבים דינמיים יכול להיות טרחני ביותר, בפרט כשרוצים להעביר פרמטרים על מנת לקבל תוצאה רצויה. שיטת הזיהוי כיום משתמשת בתכונות פיזיות של המשאב כגון, שם קובץ ופירוט מסורבל של הפרמטרים. הסגנון של REST בא לפתור את הדבר הוא בא ליצור בקשות זיהוי שמתארות את עצמן בצורה לוגית ולא פיזית על ידי תיאור צורך ולא משאב פיזי. על דרך משל: לבקש את הפטיש 5 ק"ג המונח בארון במחסן הכלים עדיף וברור יותר מלבקש את הפטיש 5 ק"ג שנמצא במחסן 6 בארון השני במדף הראשון.

דוגמא:

אם רוצים פרטי לקוח מספר 123 נפרט בקשה כזו:

http://www.my-site.co.il/customer/123

אם רוצים פרטי כל הלקוחות:

http://www.my-site.co.il/customer

אם רוצים את כל ההזמנות של לקוח 123 מחודש 03

http://www.my-site.co.il/customer/123/orders/03

תיאור כזה הוא פשוט יותר וקל לניסוח מהאלטרנטיבה הקודמת הפיזית, כמו כן הוא גם מסתיר את המשאבים הפיזיים שקיימים בשרת (שמות קבצים ונתיבים אליהם פונים).

תיאור כזה של המשאב ב – REST מחייב לפתח בצד שרת שירות אשר יקבל את הבקשה כמו שהיא, ינתח אותה וינתב אותה לגורם המתאים בשרת ולבסוף ישלח תגובה response למי שבקש את המשאב. כאמור הבקשות מתארות עכשיו את הצורך בצורה לוגית ולא את המשאב אשר עליו לבצע את המשימה. שירותים כאלה נקראים web services. כל סביבה עם הכלים שלה מממשים את סגנון העבודה הזה (node , .net , php , java וכו') הממשק אליו פונים.

https://youtu.be/BgxGBVUnSxw

לסיכום:

- בצד שרת מכינים את השירותים באוסף של web services נקרא api
- כל משאב מנוסח ומקבל זיהוי מתאים לפי סגנון עבודה זה
- ניתן לשלוח בקשות ישירות מדפדפן , על ידי AJAX על ידי כלים כמו: curl ו- postman

3. מימוש

3.1.1. יצירת ממשק API ב - PHP

דבר ראשון יוצרים את ה –API אשר יספק את השירותים. השירות במקרה שלנו הוא: הבאת רשימת עובדים כל הרשימה – דבר זה יתבטא בבקשה להבאת כל העובדים לפי הפירוט הבא:

http://localhost/Employees

השירות הנוסף יהיה להביא עובד אחד לפי מספר העובד לפי הפירוט הבא:

http://localhost/Employee/1003

ה – response יהיה בפורמט json יגיע לדפדפן אשר יעצב אותו לפי הצורך.

https://youtu.be/BAOS8nQQMM0

3.1.2. הגדרת הניתוב ב PHP

על מנת לתמוך במבנה הבקשה ב - REST יש להגדיר באפליקציה שכל בקשה תנותב לנקודה אחת מרכזית. הדבר נעשה בעזרת קובץ htaccess. קובץ זה מורה לשרת הווב apache להפנות כל בקשה ללא יוצא מהכלל ונקודה מרכזית אחת שממנה יתפתח כל השירות לבקשותיו. להלן מבנה קובץ הניתוב:

```
1 RewriteEngine On

2 RewriteCond %{REQUEST_FILENAME} !-f
3 RewriteCond %{REQUEST_FILENAME} !-d

4 RewriteRule ^(.+)$ index.php?url=$1
```

שורה מספר 1 מורה לשרת להפעיל את שירותי הניתוב מחדש
שורה מספר 2 ו – 3 מתנה כך שאין בבקשה דרישה למשאב שהוא קובץ פיזי או תיקיה פיזית
שורה מספר 4 מגדירה את כלל הניתוב שאומר כל מה שמגיע אחרי שם הדומיין הופך לפרמטר אשר נשלח

לבקשה למשאב בשם index.php, המקבל פרמטר בשם url לדוגמא:
הבקשה הזו:

http://localhost/Employee

מתחלקת כך:

http://localhost - domain
/Employee - url

ואז בתוך index.php מתבצע ניתוח של ה - url על ידי פירוקו לגורמים לפי התו (/) לאחר הפירוק
האפליקציה מפעילה את השירות לפי הפרמטרים שהגיעו כך:

```
<?php
1 require_once 'Employees.php';

2 $url = explode('/', rtrim($_GET['url'],'/'));
3 $controller = $url[0];
4 $parameter = isset($url[1])? (int)$url[1] : 0;

5 switch ($controller) {
6    case 'Register':
7        echo 'ok';
8        break;
9    case 'Employees':
10        Employees::$empno = $parameter;
11        echo Employees::getData();
12        break;
```

```
13     default:
14         header('HTTP/1.0 400 Bad Request');
15         echo 'Bad request: ' . $controller;
16 }
```

בשורות 2 עד 4 מתבצע הניתוח של ה- url ובחלקים שלו הופכים למידע אופרטיבי בתוכנית. למשל: כניסה 0 במערך מהווה controller המאפשר לזהות את השירות. בשורה 4 נבדקת האפשרות שהועבר פרמטר, במקרה הנדון זה יהיה מספר העובד (וכמובן בנקודה זו יש לבצע בדיקת תקינות שאכן לפנינו מספר למשל). נא לשים לב שאם לא הגיע פרמטר, כניסה זו תהייה ריקה והיא תקבל ערך ברירת מחדל 0. כך אפשר יהיה לממש את השירות על ידי פונקציה אחת אשר תקבל תמיד פרמטר אבל תבדוק אם ערכו 0 זה סימן שיש להחזיר את כל העובדים, ואם יהיה מספר כלשהו שונה מ – 0, המהלך יחפש עובד עם מספר זהה, ואם הוא ימצא הוא ישלח את הנתונים, אחרת תשלח הודעת שגיאה שעובד במספר כזה אינו קיים. משורה מספר 5 מתבצע ניתוח איזה מבין השירותים אמור להתבצע.

הרכיב שמבצע את עבודה נמצא ב – class שקרא Employees.php רכיב זה נקרא controller להלן פירוט הרכיב:

```php
<?php

class Employees {
    static public $empno;
    static public function getData() {
        $jdata = file_get_contents('employees.json');
        $returnData = $jdata;
        if (self::$empno != 0) {
            $assoc_data = json_decode($jdata,true);
            foreach ($assoc_data as $item) {
                if ($item['empno'] == self::$empno ){
                    $returnData = json_encode($item);
                    break;
                }
                $returnData = '{"error": "notfound"}';
            }
        }
        return $returnData;
    }
 }
```

הנתונים מאוחסנים בקובץ json :


```
[
    {
        "empno": 1000,
        "ename": "Aron",
        "sal": 7900,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1982-09-06T08:04:56.000Z",
        "deptno": 10
    },
    {
        "empno": 1001,
        "ename": "Mishko",
        "sal": 7100,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1988-09-06T08:04:56.000Z",
        "deptno": 12
    },
    {
        "empno": 1002,
        "ename": "Barbur",
        "sal": 8700,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1981-09-06T08:04:56.000Z",
        "deptno": 10
    },
    {
        "empno": 1003,
        "ename": "Eliyahu",
        "sal": 4600,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1985-09-06T08:04:56.000Z",
        "deptno": 10
    },
    {
        "empno": 1004,
        "ename": "Michel",
        "sal": 2600,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1984-09-06T08:04:56.000Z",
        "deptno": 10
    },
    {
```

```
        "empno": 1005,
        "ename": "ShuShu",
        "sal": 5200,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1987-09-06T08:04:56.000Z",
        "deptno": 20
    },
    {
        "empno": 1006,
        "ename": "Pinto",
        "sal": 8600,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1982-09-06T08:04:56.000Z",
        "deptno": 20
    },
    {
        "empno": 1007,
        "ename": "Pinchus",
        "sal": 4600,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1989-09-06T08:04:56.000Z",
        "deptno": 20
    },
    {
        "empno": 1008,
        "ename": "Shmulick",
        "sal": 3700,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1988-09-06T08:04:56.000Z",
        "deptno": 30
    },
    {
        "empno": 1009,
        "ename": "Efi",
        "sal": 3400,
        "hiredate": "1991-01-31T00:00:00.000Z",
        "dob": "1980-09-06T08:04:56.000Z",
        "deptno": 30
    }
]
```

3.1.3. יצירת ממשק API ב – NODEJS

במימוש בסביבה זו נעשה במקרה שלנו על ידי vanilla javascript כלומר ללא חבילות עזר על ידי הכלים העומדים לרשותנו בשפה, אין צורך לנתב מחדש את הבקשות והתוכנית מנתחת את ה – url ומגיבה בהתאם להלן הקוד:

```
const http = require('http');
const decide = require('./decide').decide;
const requestHandler = (request, response) => {
    console.log(request.url);
    if (decide(request.url, response)) {
        response.end('Comming soon....');
    }
}

var server = http.createServer(requestHandler);

server.listen(8080, () => {

});

console.log('Litening on port 8080 - [REST Api]');
```

מודול שנותן את השירות כתוב גם הוא ב – vanilla javascript להלן הקוד:

```
const fs = require('fs');
var res = null;
exports.decide = function(url,response) {
    res = response;
    var params = url.split('/');
    var retValue = false
    switch (params[1]) {
        case '':
            sendHomePage()
            break;
        case 'Employees':
            if (params.length > 2) {
                Employees(params[2]);
            } else {
                Employees(0);
            }
            break;
        default:
```

```
            console.log('Bad REST....')
    }
    return retValue;
}

function sendHomePage() {
    data = fs.readFileSync('homepage.html');
    res.end(data);
}

function Employees(num){
     var data = fs.readFileSync('employees.json');
     var emp = num == 0  ? 0 : null;
     var error = "{'error': 'not found'}"
     if (num) {
        emps = JSON.parse(data);

        for (let i =0 ; i < emps.length; i++) {
            if (emps[i].empno == num) {
                emp = JSON.stringify(emps[i]);
                break;
            }
        }
        data = emp == null ? error : emp;
        res.end(data);
    } else {
        res.end(data);
     }
}
```

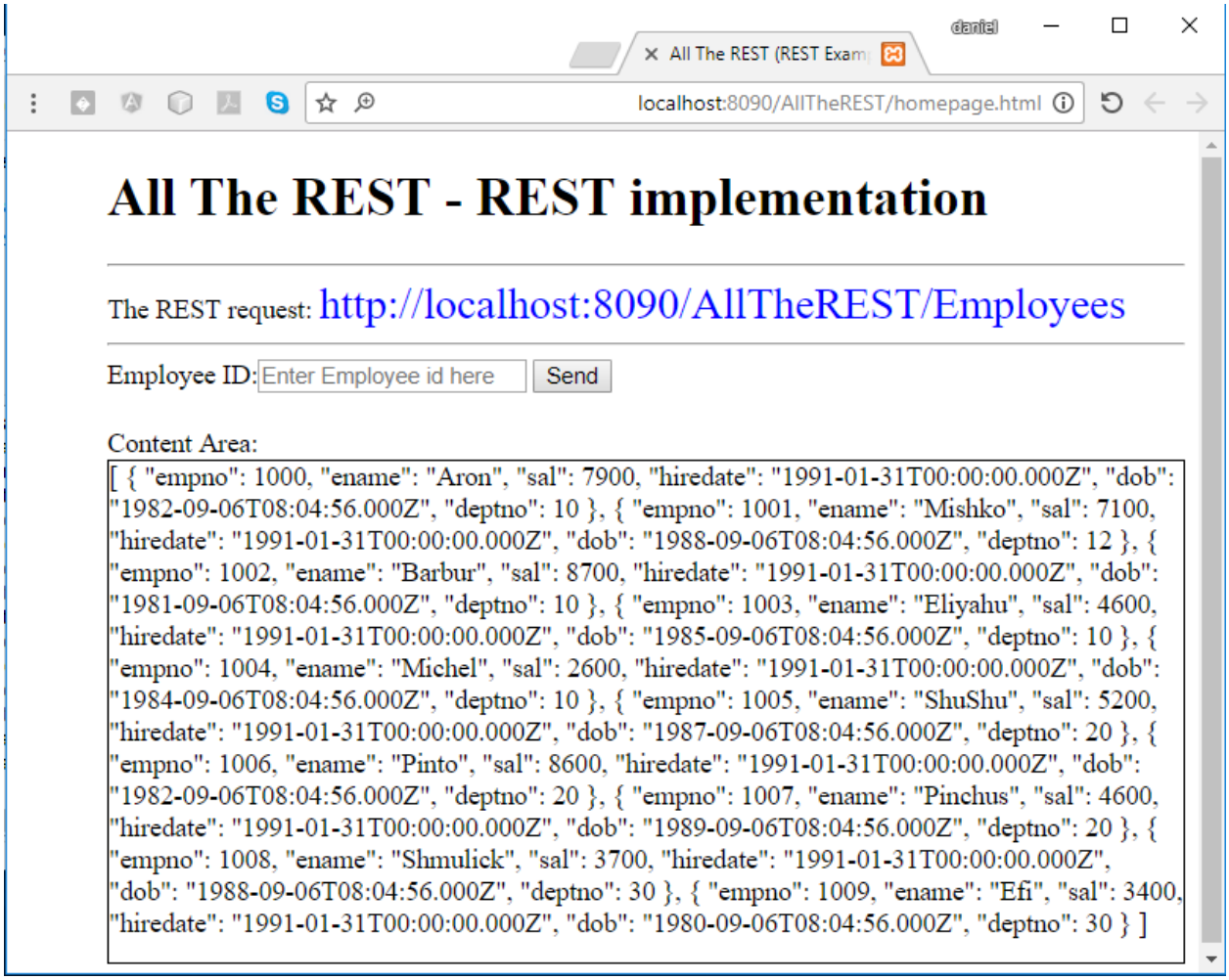
מהלך זה מפרק את ה - url ומפעיל את השירות המבוקש אשר נמצא במודול . המימוש כאמור נעשה בכלים הטבעיים של השפה ללא frameworks וכדו. מראה כמה פשוט לממש את הסגנון שמביא איתו ה – REST.

3.1.4. צד הדפדפן

על מנת לבדוק שהשירות מתפקד יש דף html אשר מאפשר לנו לשלוח בקשות לשרת עם ה - API שפורט לעיל. נעשה שימוש ב Ajax על ידי שליחת בקשות המתאימות לשרת הפעיל ומתקבל מידע ב text בפורמט json ה – script בדף מקבל את המידע יכול לנתח אותו ולעצב אותו לפי הצורך.

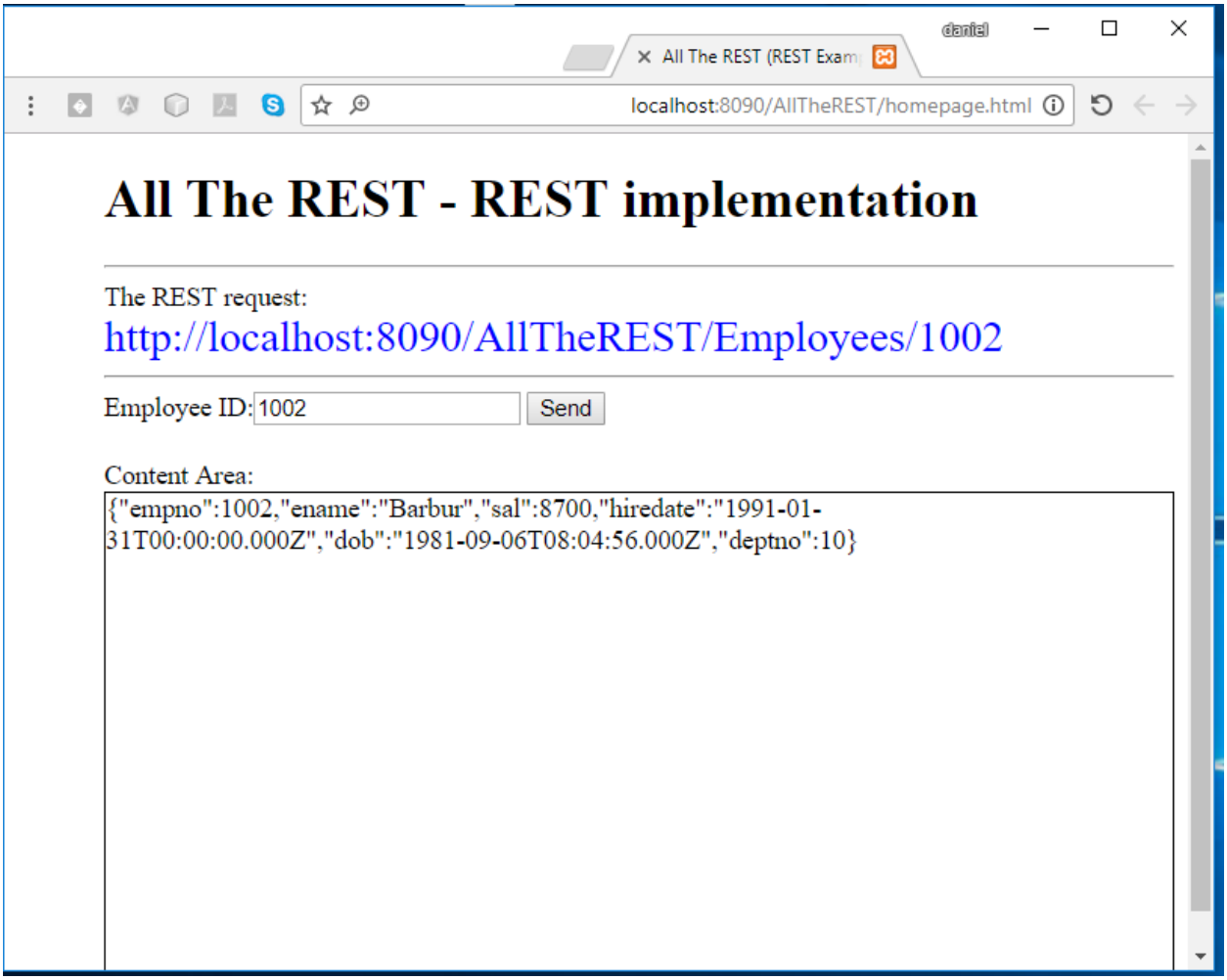
דף זה יכול לעבוד מול כל שרת אשר מצפה לקבל בקשות בסגנון REST, שאליו הוא יחזיר את התגובה.

להלן צילום מסך המתאר כיצד תראה תגובה לבקשה של כל העובדים:

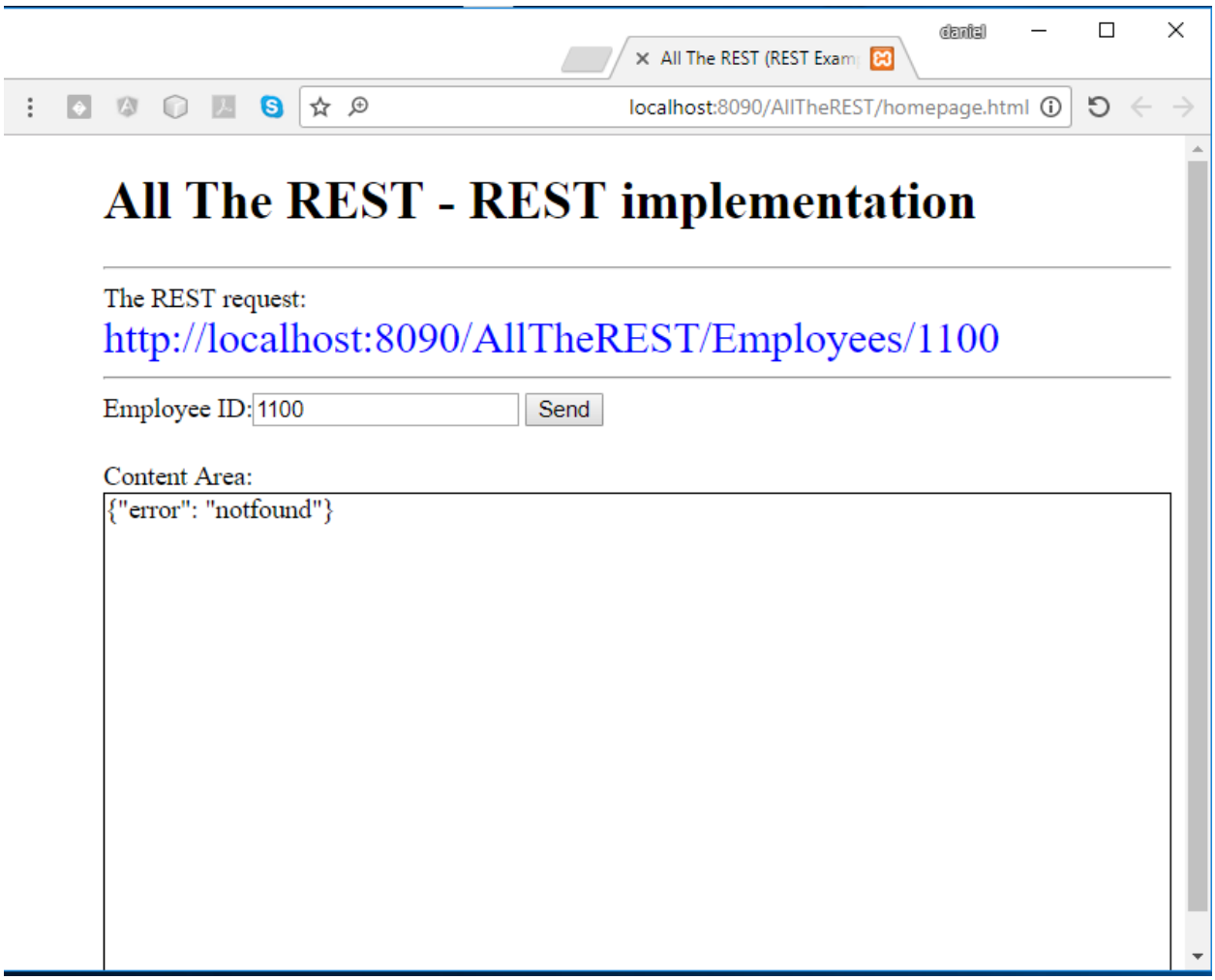


הבקשה נשלחה ב Ajax מתוארת מתחת לכותרת עם התיאור The REST request . לחיצה על כפתור send שלחה את הבקשה כאשר בתיבת הקלט Employee ID לא נרשם דבר. האפליקציה "הבינה" שבאין קלט יש ליצור בקשה לכל העובדים. התוצאה חזרה לחלון התוכן המוגדר ב – html בפורמט json מכן הכל פתוח להחלטה איך להציג את הנתונים.

המסך הבא מתאר מקרה בו מתבקש עוד מסוים לפי מספרו:



במקרה זה רק פרטי העובד מגיעים לעיבוד. וה – url מתאר את הנתונים המבוקשים מבקשים לפנות שירות אשר יספק את פרטי העובדים ומתבקש עובד מספר 1002 אם יבוקש מספר עובר שלא קיים תתקבל הודעה כמוצג בצילום מסך הבא.



מספר עובד 1100 לא קיים ולכן מופיעה הודעת שגיאה. נא לשים לב שגם השגיאה חוזרת במבנה json כך שהטיפול בהודעות שגיאה יהיה דומה לטיפול בהודעות רגילות אשר מביאות נתונים תקינים.

https://youtu.be/zZtEc46i35U

להלן הקוד שנמצא בצד הלקוח client שרץ בדפדפן:

```
<!DOCTYPE html>
<html>
<head>
    <title>All The REST (REST Example)</title>
</head>
<body style="margin-left:60px">
    <h1>All The REST - REST implementation</h1>
    <hr>
```

```
      <div>The REST request:
          <span style="color: blue;font-size: 26px"
              id="theRequest"></span></div>
    <hr>
      <label for="employeeID">Employee ID:</label><input id="EmployeeID"
                    type="text" title="BLANK for all Employees"
                    placeholder="Enter Employee id here">
    <button onclick="sendRequest()">Send</button>
    <div style="margin-top:20px">Content Area:</div>
    <div id="content" style="height: 300px; border: solid 1px black"></div>

    <script>
            var xhr;
            function sendRequest() {
                var empid = document.getElementById('EmployeeID').value;
                // PHP implementation
                var url = "http://localhost:8090/AllTheREST/Employees";
                // Node.js inplementation
                //   var url = "http://localhost:8080/Employees";
                if (empid) {
                    url += '/' + empid;
                }
                document.getElementById('theRequest').innerHTML = url;
                getData(url);
            }

            function getData(url)
            {
                xhr = new XMLHttpRequest();
                xhr.onreadystatechange = handleResponseData;
                xhr.open('GET', url);
                xhr.send();
            }
            function handleResponseData() {
             //   console.log(xhr.readyState)
                if (xhr.readyState === 4) {
                    if (xhr.status === 200) {
                        var content = document.getElementById('content');
                        content.innerHTML = xhr.responseText;
                    } else {
                        alert('Ooops There was a problem with the request.');
                    }
                }
            }
    </script>
</body>
</html>
```

קישורים לעיון:

סרטוני הדרכה

https://www.youtube.com/playlist?list=PLdZoWGdVe4Bs7OEq4_VzxEWPwuymeojSO&disable_polymer=true

Hypertext Transfer Protocol -- HTTP/1.1
https://tools.ietf.org/html/rfc2616

https://tools.ietf.org/html/rfc7230

https://tools.ietf.org/html/rfc7231

https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol

https://www.jmarshall.com/easy/http/

http://icode.co.il/%D7%A4%D7%A8%D7%95%D7%98%D7%95%D7%A7%D7%95%D7%9C-http-%D7%9E%D7%93%D7%A8%D7%99%D7%9A-%D7%9C%D7%9E%D7%AA%D7%97%D7%99%D7%9C%D7%99%D7%9D/

Components of URLs
https://www.youtube.com/watch?v=A79kP_Zjnio

https://stackoverflow.com/questions/176264/what-is-the-difference-between-a-uri-a-url-and-a-urn

REST

https://he.wikipedia.org/wiki/REST

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

https://www.infoq.com/articles/designing-restful-http-apps-roth

https://www.infoq.com/articles/rest-introduction

REST

תרגיל

יש ליצור ממשק API אשר יתן שירותי חישוב של ארבע פעולות החשבון : חיבור חיסור כפל חילוק. את הממשק אפשר לכתוב ב – PHP או ב – NODE JS (הפתרון מכיל דוגמאות לשתי האפשרויות)

הפניה לממשק תהיה לפי כללי ה – REST לדוגמא:

http://localhost:8080/add/10/10
http://localhost:8080/sub/10/10
http://localhost:8080/mul/10/10
http://localhost:8080/div/10/10

לבדיקה ניתן להשתמש בתוכניות השירות **curl** כמוצג להלן: (או בעזרת POSTMAN)