**Lesson 42-46 - RESTful Service**

# Introduction - HTTP and Web Service

**HTTP**

* כדי להעביר מידע בין שני מחשבים יש לנו שלבים קבועים אותם אנחנו צריכים לבצע

* פעולות קבועות אלו נקראות פרוטקולי תקשורת

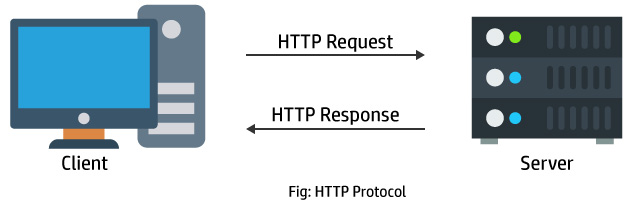
סוגים של פרוטקולי תקשורת:

* UDP – משמש בעיקר להעברת קבצי וודאו
* TCP – משמש בעיקר להעברת מידע טהור כולל בדיקות
* PTI – משמש בעיקר לעברת קבצים
* HTTP – פרוטוקול תקשורת ברשת האינטרנט

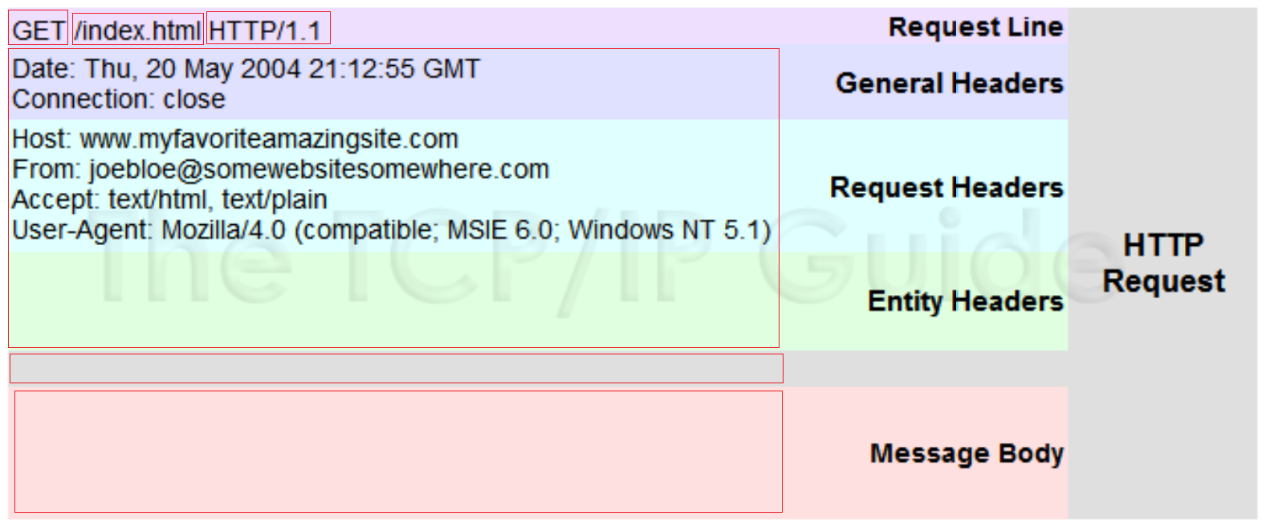
HTTP- אינו באמת פרוטוקול תקשורת - HTTP מגדיר את הפורמט בו עובר המידע. מאחורי

הקלעים פועל פרוטוקול התקשורת TCP

לפי פרוטוקול HTTP , כדי שתתרחש העברת מידע בין שני מחשבים שהתוצאה שלה היא דף אינטרנט צריכים להישלח שני מסמכים

1. Request – מהמחשב המבקש את הדף
2. Response – מהמחשב השולח את הדף (למחשב השולח דף אינטרנט אנחנו קוראים שרת)

* פרוטוקול HTTP מגדיר את הפורמט בה צריכים להיות כתובים שתי המסמכים

**מסמך request**

3

2

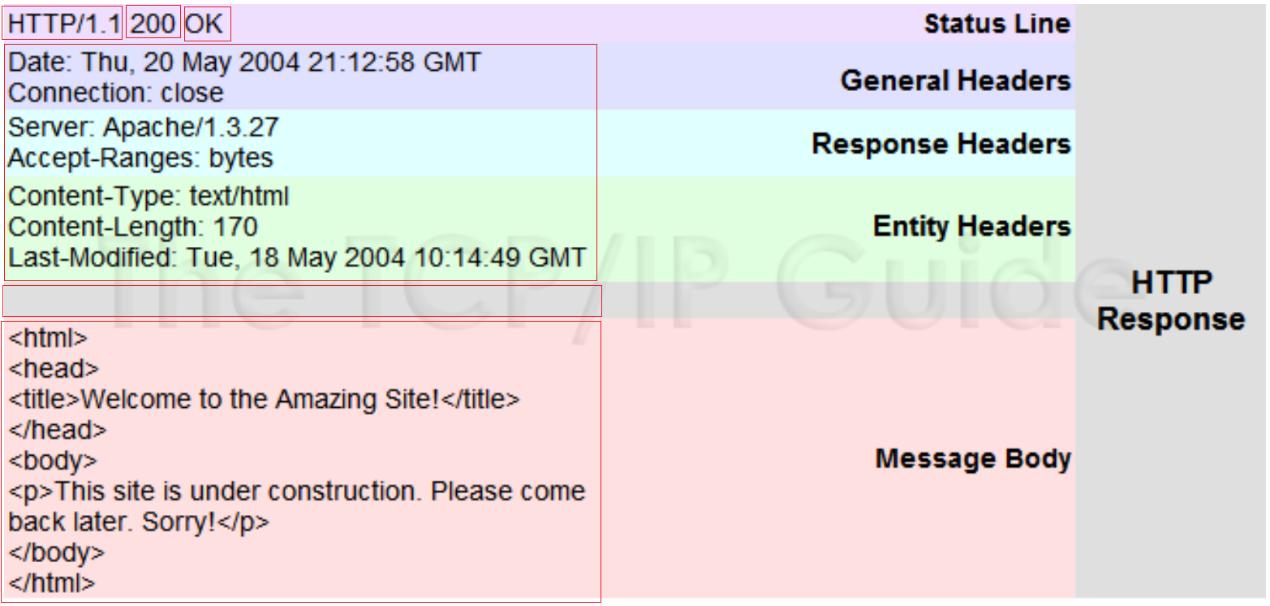
1

4

5

6

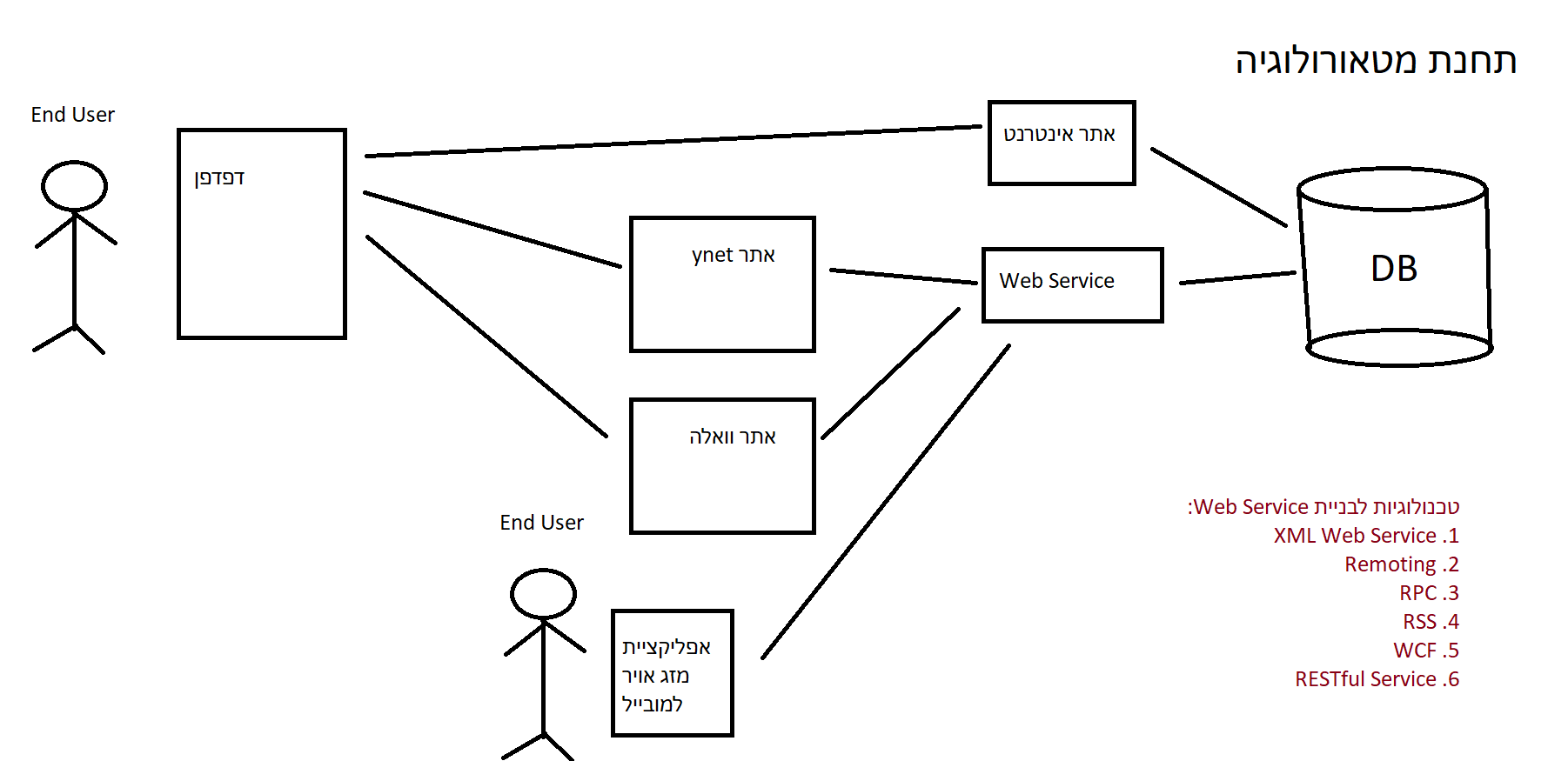
1. Method - סוג הבקשה - GET – לקבל מידע, POST – לשלוח מידע
2. Page - שם הדף המבוקש
3. Version – הגרסה של הפרוטוקול
4. Headers – מידע נלווה לבקשה – תאריך, שעה, כתובת IP
5. שורה ריקה – באה להפריד בין headers ל body
6. Body – המידע עצמו (במקרה של GET לא שולחים שום דבר)

**מסמך response**

1. Version – הגרסה של הפרוטוקול
2. Status code – קוד מספרי שנותן תשובה לגבי התהליך (2xx – הצלחה, 404 – לא נמצא וכו)
3. Status text – מידע טקסטואלי על התהליך
4. Headers – מידע נלווה על התהליך מהשרת
5. שורה ריקה
6. Body – הדף שביקשנו.

**Web Service**

* שירות שחושף מידע דרך האינטרנט

נמחיש על ידי סכמה:

1. יש לנו תחנת מטאורולוגיה שרוצה להרוויח כסף – היא רוצה לחשוף את המידע המטאורולוגי שיש לה ב DB
2. למחשב פרטי אין גישה ל DB אז איך התחנה יכולה לחשוף את המידע ?
3. פתרון – התחנה תחשוף את המידע ב DB לאתרים של חברות אחרות (שישלמו לה על המידע)
4. המנגנון שמאפשר לתחנה להחצין את המידע נקרא Web Service -

* כאשר מתכנת חיצוני רוצה לחלץ את המידע מה DB הוא יעשה זאת על ידי API - אוסף של פונקציות שמוגדרות ב web service שנותנים גישה ל DB

1. ישנם טכנולוגיות שונות לבנות Web Service ואת מהן היא REST

# REST- Re-presentational State Transfer

**איך כל זה מתקשר ביחד ?**

תשובה – REST נבנית על גבי HTTP – אנחנו לוקחים את המסמכים של HTTP (שתפקידים

להעביר דפי אינטרנט) ועושים להם אדפטציה כך שיעבירו מידע

**מה השינויים שאנחנו מבצעים ?**

ב request :

1. Method – במקום שני סוגים של פעולות יהיו לנו עכשיו 4:

* GET – מחזירה מידע
* POST – הוספת מידע
* PUT – עדכון מידע מלא
* *PATCH) – עדכון מידע חלקי - כמעט ולא בשימוש)*
* DELETE – מחיקה של מידע

[הערה – שם נוסף ל 4 הפעלות הנ"ל הוא CRUD (Create, Read, Update, Delete) ]

1. Page – שולחים route – לאיזה פונקציה מתוך ה API אנחנו רוצים לגשת
2. Body – (חוץ מ GET) – פורמט כלשהי להצגת מידע (לרוב Json)

ב response

* רק ה body משתנה - יכיל את המידע המוחזר (לרוב json)

**תרגול בכיתה**

1. כדי להרים web service נתקין Package של npm שנקרא express – npm I express
2. קריאה ל package שהתקנו

const express = require("express")

1. express היא פונקציה ולכן אנחנו צריכים לאתחל אותה?

const server = express()

* לתוך server נקבל את כל פונקציות שדרושות לנו להרים את השרת ולכתוב API

1. נרים שרת – נבחר port מסויים שעליו נעבוד (לרוב 3000 ומשהו)

server.listen(3000, () => console.log("Listening on http://localhost:3000")

* הפונקציה listen מקבלת 2 פרמטרים – 1. מספר הפורט אליו אנחנו מאזינים

2. פונקציית callback – בתוך פונקציית callback נדפיס את הכתובת של ה port שלנו כדי שנוכל לפתוח אותו בדפדפן

* כתובת ה port שלנו היא – [http://localhost:"portNumber](http://localhost:%22portNumber)" – כאשר נלחץ על הקישור ה port יפתח בדפדפן – עברנו לצד לקוח

1. כדי להציג תוכן בדף שנפתח נשתמש בפונקציה get

server.get("/", (request, response) => {

  response.send("Wolcome to our WebSite !")

})

הפונקציה מקבלת 2 פרמטרים:

1. route אליו אנחנו רוצים לגלוש (במקרה הזה דף הבית)
2. פונקציית callback – גם היא מקבלת 2 פרמטרים 1. request

2. response

* כל אחד מהם הוא אובייקט שמאפשר לנו לבצע פעולות שונות

לדוגמה send – הצגת string בדף שנפתח

**הסבר - מה בעצם מתרחש כאשר אנחנו פותחים את הדפדפן ?**

1. ברגע הפתיחה נשלחת בקשת GET לשרת שיצרנו עם ה route הנוכחי (במקרה הזה נפתח דף הבית ולכן ה route הוא /)

* כתיבה

1. השרת שיצרנו מקבל את ה request ומחפש האם יש לו פונקציה המכילה את ה route המבוקש
2. במקרה הזה – קיימת פונקציה לבקשת GET ששולחת בחזרה לדפדפן את ה string שכתבנו
3. הדפדפן מציג את ה string.

לרוב לא מציגים string אלה אובייקט json

1. נבנה אובייקט Json

  const weather = {

    description: "Cold Weather",

    degrees: 20,

    location: "Isreal"

  }

  response.json(weather)

})

1. נכתוב בקשת GET

server.get("/weather", (request, response) => {

  response.json(weather)

})

* נכתוב route אליו אנחנו רוצים להגיע (/weather)
* את המידע נציג על ידי response.json

1. על ידי request אנחנו יכולים לחלץ מידע מה router

server.get("/weather-by-city/:city", (request, response) => {

  const city = request.params.city;

  response.send(`Cold Weather in ${city}!`)

})

* הפעם ה route כתוב בצורה שונה – city צריך להיות routeדינמי - לכן נוסיף לפניו נקודתיים (city:) כדי להגיד למערכת – תפעיל את ה callbackלכל נתיב שבא אחרי weather-by-city

הסבר על הניתובים:

* הניתוב weather-by-city/city – מחייב לגלוש ל weather-by-city/city כדי להפעיל את ה callback.
* לעומת זאת weather-by-city/:city – מאפשר להפעיל את callback על כל נתב לדוגמה – weather-by-city/1 או weather-by-city/blabla וכו
* request.params.city – פקודה המאפשרת לנו לחלץ המידע של נתב :city
* עכשיו אנחנו יכולים להשתמש במידע שהוכנס – במקרה הזה נדפיס אותו למסך

# REST Convention

* מוסכמה לכתיבה של request ו response לפי ה method בהם אנחנו משתמשים ב REST:

HTTP Request

|  |  |  |  |
| --- | --- | --- | --- |
| What do we want to do? | Method | Route | Body content |
| Get All | GET | /api/cats | --- |
| Get One | GET | /api/cats/7 | --- |
| Add New | POST | /api/cats | JSON Object of the Cat to Add |
| Update Full | PUT | /api/cats/7 | JSON Object of the Full Cat to Update |
| Update Partial | PATCH | /api/cats/7 | JSON Object of the Partial Cat to Update |
| Delete Cat | DELETE | /api/cats/7 | --- |

HTTP Response

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| What do we want to do? | Success Status Code | | Success Status Text | Body content |
| Get All | 200 | OK | | JSON Array of All Objects |
| Get One | 200 | OK | | JSON Object |
| Add New | 201 | Created | | JSON Object of the Added Cat Including the ID |
| Update Full | 200 | OK | | JSON Object of the Updated Cat |
| Update Partial | 200 | OK | | JSON Object of the Updated Cat |
| Delete Cat | 204 | No Content | | --- |

* על כל שאר הבקשות שאנחנו רוצים לשלוח (לדוגמה כל החתולים לפי צבע, גיל) אין מוסכמה

דוגמאות בכיתה לכתיבת method

1. פותחים קובץ חדש, יוצרים קובץ json (npm init), מתקינים express ומאתחלים

const express = require("express");

const server = express();

1. יוצרים Json array עליו נעבוד

const products = [

    { id: 1, name: "Apple", price: 3.5 },

    { id: 2, name: "Banana", price: 5.7 },

    { id: 3, name: "Peach", price: 6.2 },

];

1. מתקינים תוכנה הנקראת Postman המאפשרת לדמות בקשות מהדפדפן
2. הוספת אובייקט ((POST

* לפי המוסכמה ה route להוספת אובייקט יחיד הוא - /api/(Object Name in plural)

server.post("/api/products", (request, response) => {

הסבר קצר לפני

* כדי שנוכל להוסיף מוצר, השרת צריך לקבל אובייקט json מה body של בקשת ה request
* הבעיה – האובייקט שנשלח בבקשת ה request ה string ולא אובייקט Json
* אנחנו צריכים להמיר את ה string שמתקבל – נעשה זאת על ידי שימוש ב middleware function – פונקציה שמופעלת לפני שהפונקציה לבקשת request מופעלת

server.use(express.json());

* use() – פונקציה מובנית מתוך express שמאפשרת לנו להוסיף middleware function (יכולים להיות הרבה use, כל אחת פעולה אחרת)
* במקרה הזה ה middleware function בה השתמשנו היא express.json() – היא ממירה את ה Data שמגיע מה request לאובייקט שנקרא body.

בחזרה להוספת המוצר –

* נכניס את ה data שבאובייקט body לתוך משתנה

    const product = request.body;

* נדמה יצירה של id

    product.id = products.length + 1

* נוסיף את המוצר למערך המוצרים

    products.push(product);

* לפי המוסכמה - נשלח בקשת response עם האובייקט החדש

    response.status(201).json(product);

* במקרה של הוספת מוצר status code ה responseמשתנה – באופן דיפולטיבי response.json() שולח 200. אנחנו צריכים לשנות את הקוד ל 201 (כיוון שזאת המוסכמה)

1. עדכון מלא של אובייקט ((PUT

* לפי המוסכמה ה route שיפעיל את הפונקציה הוא – api/(Object Name in plural)/:id
* כזכור זה אותו routeהשתמשנו גם כדי להציג מוצר בודד על ידי בקשת GET - כיוון שה method שונה אין התנגשות

server.put("/api/products/:id", (request, response) => {

* כמו במקרה הקודם גם כן אנחנו מקבלים מידע מה request ולכן נקרא ל request.body

const product = request.body;

* נקבל את ה id מתוך ה route

    const id = +request.params.id;

* על ידי ה Id נמצא את ה Object אותו אנחנו רוצים לעדכן ((productToUpdate

    const productToUpdate = products.find(p => p.id === id);

* נכניס את ה data החדש שנמצא ב product לתוך האובייקט שמצאנו

productToUpdate.name = product.name;

productToUpdate.price = product.price;

* לפי המוסכמה אנחנו צריכים לשלוח ב response את המוצר המועדכן

    response.json(productToUpdate);

1. עדכון חלקי של אובייקט ((PATCH
2. לפי המוסכמה ה route שיפעיל את הפונקציה הוא – api/(Object Name in plural)/:id

server.patch("/api/products/:id", (request, response) => {

1. גם כאן אנחנו צריכים למצוא את המוצר (לפי ה id) ולהשתמש במידע שקבלנו מה request

const id = +request.params.id;

const product = request.body;

const productToUpdate = products.find(p => p.id === id);

###### אפשר לקבוע איזה field בתוך האובייקט אחנו רוצים לעדכן

1. בכיתה בנינו עדכון בצורה דינמית:

###### לולאת for in - לכל field שקיים באובייקט שהגיע ב request

    for (const prop in product) {

###### בדוק – האם אותו field קיים גם באובייקט productToUpdate

        if (prop in productToUpdate)

###### אם כן, עדכן

            productToUpdate[prop] = product[prop];

        }

    }

* ושוב לפי המוסכמה אנחנו צריכים לשלוח ב response את המוצר המועדכן

    response.json(productToUpdate);

1. מחיקת אובייקט (DELETE)

* לפי המוסכמה ה route שיפעיל את הפונקציה הוא – api/(Object Name in plural)/:id

server.delete("/api/products/:id", (request, response) => {

* כדי למחוק את המוצר נמצא את המיקום שלו במערך על ידי הפונקציה findIndex

const id = +request.params.id;

const index = products.findIndex(p => p.id === id);

* על ידי splice נמחק אותו מהמערך

    products.splice(index, 1);

* לפי המוסכמה ה status code המוחזר ב response לבקשת delete הוא 204
* בנוסף, לפי המוסכמה לא שולחים תוכן ב body – הבעיה – ראינו שחייבים לשלוח send או Json אחרת מתקבלת שגיאה
* אפשר לכתוב - response.status(204).send() וזה יעבוד אך לא כותבים ככה
* במקום זה משתמשים בפונקציה response.sendStatus

    response.sendStatus(204); // Empty body

});

# Layers

בעולם האמיתי לא כותבים את כל הבקשות בדף ה app אלה יש חלוקה לשכבות:

* כל שכבה מכילה מידע מסויים מתוך כל המערכת
* כל שכבה מתקשרת בצורה דו כיוונית עם השכבה שאחריה ולפניה

D.B

DAL

(Date Base Layer)

BLL (Buginese logic layer)

Controller

app

Database – מכילה את ה data (מערך אובייקטים)

DAL – שכבה שמקבלת גישה ל DB

BLL – השכבה שמכילה את הפעולות (פונקציות) אותם אנחנו רוצים לבצע (הוספת מוצר,

מחיקת מוצר וכו)

Controller – שכבה שמתעסקת עם ה router – מפנה כל route לפעולה שלו

app – השכבה המרכזית

**דוגמה בכיתה**

1. פותחים קובץ חדש, יוצרים קובץ json (npm init), ומתקינים express
2. נבנה את DLL
3. נפתח תיקייה חדשה בשם data-access-layer. בנוסף כיוון שכרגע אנחנו עד לא עובדים עם db ניצור קובץ Js המכיל את ה data ומדמה db

const products = [

    { id: 1, name: "Apple", price: 3.5 },

    { id: 2, name: "Banana", price: 4.7 },

    { id: 3, name: "Peach", price: 5.2 },

];

module.exports = {

    products

};

1. נחצין את ה data כאובייקט ע"י module.exports
2. נבנה את ה BLL
3. נפתח תיקייה חדשה בשם data-access-layer וניצור קובץ js ל data אתו אנחנו רוצים לעבוד (במקרה הזה Products ולכן נקרא לקובץ (products-logic
4. נייבא את ה data מתוך DLL לתוך dal

const dal = require("../data-access-layer/db");

1. נכתוב פונקציות לפעולות שאנחנו רוצים לבצע – לדוגמה הצגת כל המוצרים

function getAllProducts() {

   return dal.products;

}

1. נחצין את הפונקציות כאובייקט

module.exports = {

    getAllProducts,

    getOneProduct

};

1. נבנה את ה controller
2. נפתח תיקייה חדשה בשם controllers וניצור קובץ js שינהל את ה data איתו אנחנו רוצים לעבוד (products-controller)
3. ב controller אנחנו רוצים לעבוד עם ה route ולכן אנחנו צריכים להשתמש ב express

const express = require("express");

1. בקובץ ה controller אנחנו לא מעוניינים להשתמש בכלל היכולות של השרת אלה רק באלה שמנהלות route.
2. לכן אין צורך להרים את כל השרת - נקרא לאובייקט שקיים ב express שאחראי על ניהול routes - Router

const router = express.Router(); // אובייקט המסוגל לתאר נתיבים של נושא מסוים אחד

1. נייבא את האובייקט שייצרנו ב BLL לתוך productsLogic

const productsLogic = require("../business-logic-layer/products logic");

נכתוב מענה לבקשות שאנחנו יכולים לקבל לדוגמה בקשת get להצגת כל המוצרים

1. שינינו את הניתוב – כיוון שבכל הבקשות יחזור הניתוב /api/products אנחנו מחליפים אותו בניתוב ראשי ( (/

router.get("/", (request, response) => {

* את ה data של המוצרים אנחנו מקבלים מאובייקט productsLogic – מאחורי הקלעים הוא ניגש ל dal שניגשת ל DB ומייבא משם את ה data.

    const products = productsLogic.getAllProducts();

1. לבסוף לפי המוסכמה אנחנו מחזירים את ה data כ json.

    response.json(products);

});

1. נחצין את router

module.exports = router;

1. נבנה את app
2. נקרא ל express ונרים שרת

const express = require("express");

const server = express();

server.listen(3000, () => console.log("Listening on http://localhost:3000"));

1. נייבא את router מ controller לתוך productsController

const productsController = require("./controllers/products-controller");

1. נפעיל middleware – המר data שמתקבל מ request לאובייקט Json
2. server.use(express.json());
3. נפעיל middleware נוסף - כל בקשה שמתקבלת המכילה את הנתב api/products תופנה אל ה controller

server.use("/api/products", productsController);

* בצורה הזו הניתוב /api/products ששימש אותנו עד עכשיו שווה ערך לכתיבה של / - כמו שהגדרנו ב controller
* על אותו עיקרון הניתוב api/products/:id ששימש אותנו עד עכשיו שווה ערך ל /:id – כמו שכתבנו ב controller