בס"ד

קורס NodeJS תשפה

HTTP Server

בnodejs קיימת חבילה בסיסית ליצירת server. הserver יאזין לבקשות HTTP שיגיעו, ינהל אותן ויחזיר תשובות.

יצירת Server

כדי ליצור server נשתמש בחבילה http המובנית של node.

יצירת הserver נעשית באמצעות הפונקציה **createServer**. הפונקציה מקבלת כפרמטר callback, שירוץ לכל בקשה שתתקבל בserver.

לאחר יצירת הserver הבסיסי, יש לקרוא לפונקציה **listen** שמעלה את הserver וגורמת לו להתחיל להאזין לבקשות. הפונקציה מקבלת את הפרמטרים הבאים:

- port
- host (אופציונלי, הdefault הוא 127.0.0.1)
- callback להרצה לאחר שהserver עלה כראוי (אופציונלי)

```
import { createServer, IncomingMessage, ServerResponse } from
'node:http';

const HOST = "127.0.0.1";
const PORT = 5000;

const server = createServer((req: IncomingMessage, res:
ServerResponse) => {
    res.end("Hello!");
});

server.listen(PORT, HOST, () => {
    console.log(`Server is listening on: http://${HOST}:${PORT}`);
});
```

לאחר הרצת הקוד, ניתן לבצע בקשות לserver בניתוב הבא:

http://127.0.0.1:500/

כל בקשה תפעיל את הcallback שנשלח כפרמטר לפונקציה createServer, ותחזור עם התשובה !Hello.

ניהול בקשות

הcallback שמועבר כפרמטר לפונקציה createServer, מקבל פרמטרים של request וresponse. הrequest מכיל את פרטי הHTTP request שהגיע לשרת, והresponse מכיל פרמטרים ופונקציות לשליחת הHTTP response.

Request

הtype של הrequest הוא IncomingMessage, והשדות שלו כוללים פרטים על הבקשה כמו URL, method וheaders. לדוגמא בקטע קוד הבא:

```
const server = createServer((req: IncomingMessage, res:
ServerResponse) => {
    const {url, method} = req;
    console.log(`[${method?.padEnd(4)}] ${url}`);

    res.end();
});
```

פלט לדוגמא:

```
[GET ] /
[GET ] /api/requests
[POST] /api/requests
[POST] /api/requests?key=val&key2=val2
[PUT ] /api/requests/12
```

קבלת הbody של הבקשה לא מתרחשת מיד עם ביצוע הcallback. היות והbody לעיתים כבד, הוא מגיע באמצעות stream – הdata מתקבל בchunks, כל chunk כולל חלק מהbody והם מגיעים אחד אחרי השני עד לסיום העברת כל הdata.

הstream הוא אובייקט שמממש את EventEmitter, וניתן לקרוא את הdata שעליו באמצעות הרשמה לevents שהוא מזמן. הevents הרלוונטיים לקבלת הdata הם:

- data – מודיע על הגעת chunk חדש. מכיל כפרמטר ראשון את הstring שהגיע.
- end – מודיע על סיום הגעת הdata.
- error – מודיע על שגיאת בקבלת הbody.

הrequest עצמו יורש מהאובייקט stream ולכן הevents יישלחו עליו:

```
let body = "";
req.on("data", (chunk: string) => {
    body += chunk;
});
req.on("error", (err: Error) => {
    console.log(`Failed getting request body: ${err}`);
});
req.on("end", () => {
    console.log(`Request body: ${body}`);
    res.end();
});
```

Response

הtype של הresponse הוא ServerResponse, ועליו מוסיפים בתהליך ניהול הבקשה פרטים כמו statusCode, headers וstatusMessage.

הresponse כולל גם פונקציות לניהול התשובה לclient, בינהן:

- res.send – שליחת תשובה לclient
- res.setHeader - הוספת header לresponse
- res.end – סיום הטיפול בבקשה

לדוגמא בקטע הקוד הבא:

```
res.statusCode = 200;
res.setHeader('Content-Type', 'application/json');

res.end(JSON.stringify({ data: "SUCCESS", total: 1 }));
```

ניהול בקשות מתקדם

ניהול בקשות בserver, יכלול בדרך כלל אבחנה בין ניתובים שונים, ובין HTTP methods. כדי לתמוך בroutingב server שהרמנו, נצטרך לבדוק את הפרמטרים URL וmethod שעל הrequest ולנהל את הבקשה בהתאם.

כמו כן, לעיתים קרובות נרצה לקבל את הparameters מהבקשה – route parameters ו query parameters. לצורך כך נבדוק את הURL שעל הבקשה, כולל parsing וstring manipulation כדי לחלץ את הפרמטרים.

לדוגמא בקטע הקוד הבא, ישנה תמיכה בשני routes שונים, וחילוץ של הroute parameter מתוך הURL:

```
const {url, method} = req;

if (method == "GET" && url == "/api/names") {
    res.setHeader('Content-Type', 'application/json');
    res.end(
        JSON.stringify(["Sara", "Rut", "Elisheva"])
    );
} else if (method == "POST" && url?.startsWith("/api/books/")) {
    const urlParts = url.split("/");
    const bookId = urlParts[urlParts.length - 1];
    res.end(`Book ${bookId} added successfully`);
} else {
    res.statusCode = 404;
    res.end();
}
```

כפי שניתן לראות, יצירת server שמכיל מספר ניתובים, ניהול פרמטרים והחזרת תשובות בהתאם באמצעות החבילה http המובנית בnode, היא מורכבת מאד. ככל ונרצה להוסיף APIs חדשים, הקוד יהפוך למסובך יותר ויותר, ויכולת ההבנה והתחזוק שלו תרד מאד.

לכן, בשביל פרויקטים הכוללים server ברמה מתקדמת יותר, נשתמש בחבילות חיצוניות כמו express או nest שמפשטות את הקוד כך שקל ונח לכתוב, לתחזק ולהרחיב אותו.