בס"ד

קורס NodeJS תשפה

Event Emitter

הEvent Emitter בnodejs מאפשר לנהל events – להירשם אליהם ולהפעיל אותם. ניהול events עשוי להיות שימושי בעיקר בכתיבת קוד אסינכרוני, כאשר יש צורך בהרצת קוד מסוים כשמשהו קורה.

לדוגמא, כאשר מתקבלת הודעה על משתמש חדש במערכת, לעיתים נרצה שקוד אחר שרץ במקביל יעודכן על כך בזמן אמת וידע למשוך את הנתונים של המשתמש החדש.

שימושים נפוצים:

- רישום לeventsשמופעלים ע"י חבילות חיצוניות – ישנן ספריות שמשתמשות בevents כדי להתריע על שגיאה בחיבור, הודעה חדשה שהתקבלה ועוד. במקרים רבים רישום להודעות האלו וניהולן הוא נחוץ כדי שהשימוש בחבילה יהיה יעיל.
- ניהול queues עם producer וconsumers – התרעה על element חדש בqueue, על ריקון מלא שלו או על שגיאות תישלח בעזרת event ע"י הproducer. הconsumers נרשמים לקבלת הevents האלו ומנהלים אותם.
- עבודה עם sockets – שליחת events בזמן אמת למערכות שונות (ובעיקר לFE) נעשית בעזרת socket, שממומש בnodejs על גבי event emitter כך שניתן לשלוח ולקבל events.
- שימוש בdata stream – שליחה וקבלה של data גדול על גבי הnetwork נעשית באמצעות stream. ניהול של הdata של הstream אפשרי באמצעות events שמודיעים על start, על data חדש, על end ועל errors.

יצירת Event Emitter

```
import EventEmitter from 'node:events';

const eventEmitter = new EventEmitter();
```

ניתן גם ליצור class היורש מEventEmitter ולהפעיל עליו ישירות את הevents:

```
class Queue extends EventEmitter {

}
```

הרשמה לevents

```
eventEmitter.on('newMessage', () => {
    console.log('New message arrived');
});
```

הפונקציה on משמשת להגדרת event והרשמה אליו. היא מקבלת כפרמטר ראשון את שם הevent, וכפרמטר שני פונקציית callback שמתבצעת כשהevent מופעל.

ניתן להגדיר לcallback פרמטרים, כך שבהפעלת הevent הם יישלחו ויתקבלו ע"י הcallback שנרשם על הevent. לדוגמא:

```
eventEmitter.on('newMessage', (message: string) => {
    console.log(`New message arrived: ${message}`);
});
```

ניתן להעביר פרמטר אחד או יותר.

הפעלת event

```
eventEmitter.emit('newMessage');
```

או עם פרמטרים:

```
eventEmitter.emit('newMessage', message);
```

דוגמת קוד מלאה

קטע הקוד הבא כולל queue המנוהל בעזרת events. כל הודעה חדשה שנכנסת לqueue מפעילה event, והconsumer שנרשם לevent מטפל בה.

```
import { EventEmitter } from 'events';

class Queue<T> extends EventEmitter {
    private items: T[] = [];

    enqueue(item: T): void {
        this.items.push(item);
        this.emit('itemAdded');
    }

    dequeue(): T | undefined {
        return this.items.shift();
    }
}

class Producer {
    constructor(private queue: Queue<string>) {}

    startProducing(): void {
        let count = 0;
        const interval = setInterval(() => {
            count++;

            console.log(`Adding task ${count}`)
            this.queue.enqueue(`task ${count}`);

            if (count === 5) {
                clearInterval(interval);
            }
        }, 1000);
    }
}

class Consumer {
    constructor(private queue: Queue<string>) {
        this.queue.on('itemAdded', () => {
            this.processItem();
        });
    }

    private processItem(): void {
        const item = this.queue.dequeue();
        console.log(`Handling ${item}`);
```

```
    }
}

const queue = new Queue<string>();

const producer = new Producer(queue);
const consumer = new Consumer(queue);

producer.startProducing();
```