Asinhrono programiranje

Kod koji smo pisali do sada izvrsava se sinhrono - linija po linija. Dok se prethodna linija ne zavrsi, naredna nece poceti. Javascript, medjutim, ima mogucnost da ne radi na ovaj nacin. Dakle, neki kod se “moze poslati” negdje u pozadinu, a paraleleno sa tim izvrsavati neki drugi kod.

Vec smo vidjeli da JS izuzetno dobro radi sa funkcijama. Druge funkcije mozemo pozivati iz neke funkcije, funkciju mozemo definisati u okviru neke funkcije, funkciju mozemo prosletiti kao parametar nekoj funkciji itd. Ovo poslednje zapravo nosi zanimljive karakteristike koje mozemo iskoristiti za implementaciju callbacks - funkcija koja se prosledjuje i izvrsava u nekoj drugoj funkciji. Uzmimo primjer:

```
function outputFunction(input) {
        console.log(input);
}

function calculator(param1, param2, callback) {
  const sum = param1 + param2;
  callback(sum);
}

calculator(5, 5, outputFunction);
```

Prethodni primjer koristi samo jednu funkciju, ali mogli bismo imati i vise funkcija koje rade slicnu ili razlicitu stvar kao outputFunction, the koristiti je po potrebi.

No najveca moc ovakvog pistupa je kod operacija gdje se mora cekati na izvrsenje neke operacije - na primjer otvaranje fajlova, postavljanje kasnjenja na izvrsavanje neke operacije ili slicno.

Uzmimo na primjer kasnjenje (delay). JS ima nekoliko ugradjenih metoda, od kojih se najcesce koristi setTimeout (kasnjenje) i setInterval (ponavljanje sa razmakom).

Recimo da imamo funkciju sendLogs koja treba da se izvrsi na svakih 10 sekundi. Apsktraktnu implementaciju mozemo napisati na sledeci nacin:

```
function sendLogs(logs) {
        const logger = Logger.getInstance();
        logger.send(logs);
}
```

```
setInterval(sendLogs(logs), 10000);
```

Promises

Promisi su asinhroni koncept u JS koji omogucava “naknadno” izvrsavanje koda - tj. izvrsavanje u buducnosti. Promis kao sto mu ime kaze - obecava da ce se nesto izvrsiti, a da li ce to biti uspjesno ili ne, tj. sta ce biti rezultat, trenutno ne znamo.

Na primjer:

```
const promise = new Promise(function(resolveFunction, rejectFunction) {
   // Ovdje ide kod koji ce potrajati neko vrijeme
   resolveFunction("OK"); // ova funkcija se izvrsava ako je sve ok
   rejectFunction("Error");  // ova funkcija se izvrsava ako dodje do problema
});

   // ovo je kod koji konzumira promise i mora cekati na rezultat
promise.then(
   function(value) {
      console.log(value)
      // ovaj kod ce se izvrsiti ako je promise uspio
   },
   function(error) {
      console.log(error)
      // ovaj kod ce se izvrsiti u slucaju greske
   }
);

console.log("I'm still the number one!");
```

Posto je promise asinhroni, a console.log sinhrona operacija, mozemo dacunati da ce se poslednji console.log izvrsiti prije callback-a u then bloku.

Primjer sa citanjem fajla:

```
import fs from "fs";
const promise2 = new Promise(function(resolve, reject) {
   try {
     const file = fs.readFileSync("text.txt");
     if (file) {
       resolve(file.toString());
```

```
    }
  } catch(e) {
     reject("File not Found");
    }

});

promise2.then(
  function(value) { console.log(value); },
  function(error) { console.log(error); }
);
```

Async & await

async i await su moderni nacin pisanja asinhronog koda. Sustinski, iza ovih temina i dalje se kriju promisi, ali kod je generalno pregledniji i laksi za pracenje i razumijevanje.

koristeci kljucnu rijec async u sustini kreiramo Promise. Na primjer:

```
async myAsyincFunction () {
        // neki kod koji ce potrajati
}
```

Sa await cekamo da se promise “ispuni” tj. vrati rezultat. Bitno je napomenuti da se await moze izvrsiti samo u async funkciji (od ECMASCRIPT2022 moguce je i top level await, dakle van async funkcije. Na primjer:

Primjer citanja fajla upotrebom async funkcije:

```
async function readTestFile() {
const promise2 = new Promise(function(resolve, reject) {
   try {
   const file = fs.readFileSync("text.txt");
     if (file) {
       resolve(file.toString());
     }
   } catch(e) {
      reject("File not Found");
     }

});

 const res = await promise2;
```

```
  console.log(res);
}

readTestFile();
```

Testirati primjere:

1.

```
console.log(“A”);

setTimeout(() => { console.log(“B”); }, 0);

console.log(“C”);
```

2.

```
setTimeout(() => { console.log(“A”); }, 0);

let i = 0;
while (i < 10_000_000) {
        i++;
}

setTimeout(() => { console.log(“B”); }, 0);
```

Zadatak:

Upotrijebiti axios npm dependency (https://axios-http.com) kako bismo ucitali sadrzaj sa mreze na ovom linku: https://jsonplaceholder.typicode.com/posts, zatim konvertovati dobijeni json i snimiti u fajl pod nazivom “posts.json”.