Artefakt for producer / consumer - hybrid koden

```
const fs = require('fs');
const readCity = require('../common/readCity');
const locationsFromText = require('./locactionsFromText');
const extractors = require('./extract');
const _cliProgress = require('cli-progress');

let allFiles = fs.readdirSync('./zipfiles/');

const os = require('os'),
   cpuCount = os.cpus().length;


let removeUnknown = (cityNames, bookLocations) => {
   let cleaned = [];

   for (let bookLocation of bookLocations) {
       let id = cityNames[bookLocation[1]]
       if (typeof id !== "undefined") {
           cleaned.push({
               index: bookLocation[0],
               cityIndex: id
           })
       }
   }

   return cleaned;
}


class Schedule {
   constructor(jobs, parallel) {
       this.jobs = jobs.reverse();
       this.noJobs;
       this.parallel = parallel;
       this.bar = new _cliProgress.Bar({}, _cliProgress.Presets.shades_classic);


       this.worker = this.worker.bind(this)
       this.start = this.start.bind(this)
   }

   async worker() {
       while (this.jobs.length > 0) {
```

```
            this.bar.update(this.noJobs - this.jobs.length)
            let jobMetadata = this.jobs.pop();
            await this.createJob(jobMetadata)
        }
    }

    async start() {
        this.bar.start(this.jobs.length, 0)
        this.noJobs = this.jobs.length;
        let workers = []
        for (let index = 0; index < this.parallel; index++) {
            workers.push(this.worker())
        }
        await Promise.all(workers)
        this.bar.update(this.noJobs)
        this.bar.stop();
    }
}


(async () => {

    let bookAndCities = {};
    let { names: cities } = await readCity("cities15000.txt");
    let errors = ""

    let producer = new Schedule(allFiles, cpuCount)

    producer.createJob = async filename => {

        try {
            let fileContent = await new Promise(
                resolve => fs.readFile(__dirname + '/../zipfiles/' + filename,
function (err, data) {
                    resolve(data.toString());
                })
            )
            fileContent = extractors.removeFooter(fileContent);

            await new Promise(
                resolve => fs.writeFile(__dirname + '/../zipfiles/' + filename,
fileContent, resolve)
            )
            let bookLocation = removeUnknown(cities, await
locationsFromText(filename));
            try {
```

```
                let smalltext = extractors.take25lines(fileContent)

                let Part = extractors.extractPart(smalltext);
                bookAndCities[filename] = {
                    Part,
                    Authorname: extractors.extractAuthorName(smalltext),
                    Title: extractors.extractTitle(smalltext, Part),
                    cities: bookLocation
                };
            } catch (error) {
                bookAndCities[filename] = {
                    error: "Error",
                    cities: bookLocation
                };
            }

        } catch (e) {
            errors += filename +"\n"+ e.toString()+"\n"+"\n";
        }

        if (Math.random() > .9) {
            fs.writeFileSync('./booksAndCities.json', JSON.stringify(bookAndCities),
'utf8');
            fs.writeFileSync('./errors.json', JSON.stringify(errors), 'utf8');
        }
    }

    await producer.start()
    fs.writeFileSync('./booksAndCities.json', JSON.stringify(bookAndCities),
'utf8');
    fs.writeFileSync('./errors.json', JSON.stringify(errors), 'utf8');

    process.exit();
})();
```