CourseHunterDownloaScript Analysis

https://github.com/alekseylovchikov/ch-download

node --inspect-brk index.js -u https://coursehunters.net/course/typescript-masterclass-todd-motto -d /home/laczovics/Desktop/kurzusok/CoursHunterDownloadModule

1. **validateParams** from the terminal arguments
2. Check new github version to notify user (**github-version-checker**)
3. startDownloading script
3.1 Extract the userData from the argument terminals
3.2 Fetch the token with an http-request
4. Run **getVideos**
4.1 Create folder (**path**,**resolve**/**sep**/**isAbsolute**/fs.**mkdirSync**)
4.2 Create a logger, file with writeable stream. ()
    - Saving files, with **path.sep()** for different OS.
    - Create writeable stream, to dynamically inject values.

4.2 scrap data with cheerio, and insert token with **request.jar()**
    - request.jar → to insert cookies to every request
    - **cheerio** to scrap data
    - promise, to wrap all of this call In a promise

5. **Download videos**
5.0 check if it has been downloaded
    - Will read the log file with fs.**readFileSync**()
    - If the video is in the logFile, and the video exists in the file location, it is considered as downloaded completely.

5.1 downloadOneVideo(progress/progress events/ **pipes** / **writeStream** / **stdOut** / recursion)
    - **Progess** → is used to track all of the state properties, with progress stateEvents
    - **writeWaitingInfo** –> is using the data received from progress, and writes to the stdOut which is the terminal
    - **cleanLine** –> will clean the last line of the stdout.
    - Writing to logger stream + using recursion

1. **validateParams** from the terminal arguments

```
/**
 * Vaidate the parameters recieved from the process.argv CLI.
 *
 * @flags {object[]} It will be an object, containing all of the possible parameter
options as a string[]
 * @indexUrlFlag {Number} The index number of the url parameter( -u, --url) from the
process.argv array.
 * @indexDirFlag {Number} The index number of the url parameter( -d, --dir) from the
process.argv array.
 *
 * @returns {void}
 *  If the given parameters recieved in the CLI are not correct it will display again
the usage and exit the process
 *  If specific lessons were given it validate if they are correct or not.
 *
 *  Criterias:
 *    1.--> The first param has to be the url of the course
 *    2.--> The next process argument parameter should't be an actually an other flag!
 */
function validateParams(flags, indexUrlFlag, indexDirFlag) {
  const indexLessonsFlag = getFlagIndex(flags.lessons);
  if (indexUrlFlag == -1 || indexUrlFlag === process.argv.length - 1 ||
      indexDirFlag === process.argv.length - 1 ||
      indexLessonsFlag === process.argv.length - 1 ||
      isNextFlag(flags, process.argv[indexUrlFlag + 1]) ||
      isNextFlag(flags, process.argv[indexDirFlag + 1]) ||
      isNextFlag(flags, process.argv[indexLessonsFlag + 1]))
        printUsage();
  if (indexLessonsFlag !== -1) {
    validateLessons(process.argv[indexLessonsFlag + 1]);
  }
}
```

2. Check new github version to notify user (**github-version-checker)**

```
function checkNewVersion(startDownloading) {
  const options = {
    repo: 'alekseylovchikov/ch-download',
    currentVersion: pkg.version,
    includePreReleases: true
  };
  versionCheck(options, function (update, error) {
    if (error) throw error;
    if (update) {
      console.log('An update is available!');
      console.log('New version: ' + update.tag_name);
      console.log('Details here: ' + update.html_url);
    }
    console.log('Starting app...\n');
    startDownloading();
  });
}
```

3. startDownloading script →
3.1 Extract the userData from the argument terminals
3.2 Fetch the token with an http-request

```
*     1. Extract the userData from the argument terminals
*     2. fetch the token with an http-request
* */
function startDownloading() {
  // get email password
  const e = process.argv.indexOf('-e');
  const p = process.argv.indexOf('-p');
  if (e > -1 && p > -1) {
    // with email and password
    const email = process.argv[e + 1];
    const password = process.argv[p + 1];
    getToken(email, password)
      .then(token => runGetVideos(token))
      .catch(err => console.log('Check your email or password'.red));
  } else {
    // without email and password
    runGetVideos();
  }
}
```

4. **Run getVideos**

4.1 Create folder (path,resolve/sep/isAbsolute/fs.mkdirSync)

```
'use strict';
const path = require('path');
const fs = require('fs');
/**
 * Vaidate the parameters recieved from the process.argv CLI.
 *
 * @downloadFolder {string} The index number of the url parameter( -u, --url) from the
process.argv array.
 *
 * @returns {void}
 *    Will make a directory, where the videos can be stored, based on the input
directory url as string.
 *
 *      path.sep --> Used to seperate the path segments, both in Unix, windows
accoriding the OS system separator.
 *      path.isAbsolute --> Will check if the given path is absolute or not.
 *      path.resolve --> will concatenate, the seperate names, to a folder path, from
"","donwload" will be --> fullPath/download
 *
 *      So either by simply looping once, it will try to make a directory with
mkdirSync. If it is not possible, it will try with the accumulated string with
path.resolve()
 *
```

```
 */
function createFolder(downloadFolder) {
  const sep = path.sep;
```

```
  const initDir = path.isAbsolute(downloadFolder) ? sep : '';
  downloadFolder.split(sep).reduce((parentDir, childDir) => {
    const curDir = path.resolve(parentDir, childDir);
    try {
      fs.mkdirSync(curDir);
    } catch (err) {
      if (err.code !== 'EEXIST') {
        throw err;
      }
      if (curDir == makeDownloadFolderPath(downloadFolder))
        console.log(`Directory ${curDir} already exists`.blue);
      return curDir;
    }
    console.log(`Directory ${curDir} created`.blue);
    return curDir;
  }, initDir);
}
function makeDownloadFolderPath(downloadFolder) {
  const sep = path.sep;
  const folders = process.argv[1].split(sep);
  folders.pop();
  folders.push(downloadFolder);
  const downloadFolderPath = folders.join(sep);
  return downloadFolderPath;
}
module.exports = createFolder;
```

4.2 Create a logger, file with writeable stream. ()

- Saving files, with path.sep() for different OS.
- Create writeable stream, to dynamically inject values.

```
'use strict';
const path = require('path');
const fs = require('fs');
/**
 * Vaidate the parameters recieved from the process.argv CLI.
 *
 * @downloadFolder {string} The index number of the url parameter( -u, --url) from the
process.argv array.
 *
 * @returns {writeAbleStream}
 *
 * It will create a logger file, and return a writeable Stream!
 *
 */
function createLogger(downloadFolder) {
  const logFile =`${downloadFolder}${path.sep}videos.txt`
  fs.existsSync(logFile) ?
    console.log(`File ${logFile} already exists`.blue) :
    console.log(`File ${logFile} created`.blue);
  const logger = fs.createWriteStream(logFile, { flags: 'a' });
  return logger;
}
module.exports = createLogger;
```

4.2 scrap data with cheerio, and insert token with request.jar()

- request.jar → to insert cookies to every request
- cheerio to scrap data
- promise, to wrap all of this call In a promise

```
const cheerio = require('cheerio');
const request = require('request');
/**
 * Vaidate the parameters recieved from the process.argv CLI.
 *
 * @url {string} The url of the corse, to be downloaded
 * @token {string} The token to be used.
 *
 * @returns {promise}
 *
 *  request.jar() ---> will save cookie for future use.
 *
 */
function getVideos(url, token) {
  // 1. creating the request option, with inserted cookie
  return new Promise(function(resolve, reject) {
    let result = [];
    let names = [];
    const options = { url: url };
    if (token) {
      let jar = request.jar();
      const cookie = request.cookie('accessToken=' + token);
      jar.setCookie(cookie, url);
      options.jar = jar;
    }
    // Will return the results,names of the lessons.
    request(options, function(err, res, html) {
      if (!err) {
        let $ = cheerio.load(html);
        $('#lessons-list').filter(function() {
          let data = $(this);
          const dataArray = data
            .children()
            .children()
            .toArray();
          const filterData = dataArray.filter(
            el => el.name === 'link' && el.attribs.itemprop === 'contentUrl'
          );
          const filterSpan = dataArray.filter(el => el.name === 'span');
          filterSpan.map(el => {
            if (el.name === 'span') {
              const videoName = el.children[0].data.replace(/[\/:*?"<>|]/g, '');
              names.push(videoName);
            }
          });
          filterData.map(el => {
            result.push(el.attribs.href);
          });
          resolve({ result, names });
        });
      } else {
        reject(err);
      }
    });
  });
}
module.exports = getVideos;
```

5. Download videos

5.0 check if it has been downloaded

- Will read the log file with **fs.readFileSync()**
- If the video is in the logFile, and the video exists in the file location, it is considered as downloaded completely.

```
const path = require('path');
const fs = require('fs');
function findNotExistingVideo(videos, downloadFolder)
{
  let i = 0;
  for (let video of videos) {
    let filename = `${downloadFolder}${path.sep}${video.name}.mp4`;
    if (fs.existsSync(filename) && isCompletelyDownloaded(video.name, downloadFolder)) {
      console.log(`File \'${video.name}\' already exists`.blue);
      i++;
    } else {
      break ;
    }
  }
  return i;
}
function isCompletelyDownloaded(videoName, downloadFolder) {
  const downloadedVideos = findDownloadedVideos(downloadFolder);
  if (typeof downloadedVideos === 'undefined' || downloadedVideos.length === 0) {
    return true;
  }
  for (let downloadedVideoName of downloadedVideos) {
    if (videoName === downloadedVideoName)
      return true;
  }
  return false;
}
function findDownloadedVideos(downloadFolder) {
  const logFile =`${downloadFolder}${path.sep}videos.txt`;
  if (!fs.existsSync(logFile)) return [];
  return fs.readFileSync(logFile).toString().split("\n");
}
module.exports = findNotExistingVideo;
```

5.1 downloadOneVideo(progress/progress events/ pipes / writeStream / stdOut / recursion)

- Progess → is used to track all of the state properties, with progress **stateEvents**
- writeWaitingInfo –y> is using the data received from progress, and writes to the stdOut which is the terminal
- cleanLine –> will clean the last line of the stdout.
- Writing to logger stream + using recursion

```
// 1. Using progress, to access different metrics when downloading.
// Progress has 'progress' / 'error' / 'end' event listeners,
// Simply you are making a request to the videoUrl.
//
/**
 * Will log all of the data made in the request, and will create a writing stream, with
pipe, from the recieved data to create an mp4 file.
 *
 * @logger {WriteableStream} Where all of the logging detail can be stored
 * @video   {url : string, name: string }  object of the video data
 * @downloadFolder {string} where the videos will be donwnloaded
 * @nextVideo  {function} recursive function, to call itself, will return when it is
finished,
 *
 * @returns {boolean}
 *    progress --> to track downloading metrics
 *    writeWaitingInfo --> Will write the progress of the donwload to the proces.stdOut
(which is the terminal in this case)
 *    cleanLine -->        Will clear the process.stdrOutput line, and will move the
cursor
 */
function downloadOneVideo(logger, downloadFolder, video, nextVideo) {
  let videoName = video.name.replace("Урок ", "");
  console.log(`Start download video: ${videoName}`.blue);
  progress(request(video.url), { throttle: 2000, delay: 1000 })
    .on('progress', function(state) {
      writeWaitingInfo(state);
    })
    .on('error', function(err) {
      console.log(`${err}`.red);
    })
    .on('end', function() {
      cleanLine();
      console.log(`End download video ${videoName}`.green);
      logger.write(`${videoName}\n`);
      nextVideo();
    })
    .pipe(fs.createWriteStream(`${downloadFolder}${path.sep}${videoName}.mp4`));
}
module.exports = downloadVideos;

function writeWaitingInfo(state) {
  cleanLine();
  const percent = (state.percent * 100).toFixed(2),
        transferred = formatBytes(state.size.transferred),
        total = formatBytes(state.size.total),
        remaining = secondsToHms(state.time.remaining),
        speed = formatBytes(state.speed),
        text = `${percent}% | ${transferred} / ${total} | ${speed}/sec | ${remaining}`;
  process.stdout.write(text);
}
```