ლაბორატორიული 7

1. **npm init, npm install, scripts**

**init**

პროექტის ინიციალიზაციისთვის საჭიროა კონსოლში შემდეგი ბრძანების შესრულება:

>npm init

მას შემდეგ რაც რამდენიმე კითხვას ვუპასუხებთ, პროექტის ფოლდერში შეიქმნება ფაილი packages.json დაახლოებით შემდეგი შიგთავსით:

{

"name": "js-api-test",

"version": "1.0.0",

"description": "",

"main": "index.js",

"repository": {

"type": "git",

"url": "lab7"

},

"author": "gdznela",

"license": "MIT",

"dependencies": {

},

"devDependencies": {

}

}

ამ ფაილში მოთავსბულია ყველა ინფორმაცია რაც ჭირდება npm-ს მუშაობისთვის.

**install**

რაიმე ახალი js მოდულის დასაყენებლად გამოიყენება ბრძანება npm install, რომელიც ასევე ავტომატურად აყენებს მოდულის მუშაობისთვის ყველა სხვა დანარჩენ საჭირო მოდულს (dependency-ებს). მაგალითად ajax-თან მუშაობისთვის axios-ის დასაყენებლად საჭიროა შემდეგი ბრძანების შესრულება:

>npm install axios --save

--save ოპციის გადაცემა საჭიროა რათა მოხდეს package.json-ში დაფიქსირება რომ ეს მოდული ესაჭიროება პროექტს, ასევე ინახება მისი ვერსია (ჩაემატება dependencies-ში). ბრძანების შესრულების შედეგად ასევე შეიქმნება ფოლდერი node\_modules სადაც ინახება ყველა დაყენებული მოდულის ფაილი.

როდესაც ვაკომიტებთ კოდს source control-ში უმეტეს შემთხვევაში ვაიგნორებთ node\_modules ფოლდერს(.gitignore). ამიტომ source control-დან ახალ აღებულ პროექტში საჭირო მოდულები არ არის დაყენებული სანამ არ გავუშვებთ ბრძანებას npm install პარამეტრების გარეშე. ბრძანების შესრულების შედეგად მოხდება packages.json წაკითხვა და ყველა აუცილებელი მოდულის დაყენება რომელიც მითითებულია მასში.

ხანდახან ზოგი მოდული საჭიროა მხოლოდ დეველოპმენტის დროს, მაგალითად js bundling ინსტრუმენტები როგორიცაა browserify. ასეთი მოდულების დაყენებისას მიეთითება ოპცია --save-dev. მაგალითად browserify-ს დასაყენებლად:

>npm install browserify --save-dev

შედეგად browserify დაემატება package.json-ში devDepenendcies-ში.

რაიმე მოდულის გლობალურად დასაყნეებლად (არა მარტო ლოკალურ ფოლდერში არამედ ნებისმიერ ადგილას რომ ხელმისაწვდომი იყოს კონსოლში) გამოიყენება -g ოპცია, მაგალითად:

npm install browserify -g

**scripts**

ხანდახან პროექტის დაბილდვისთვის ან სხვა ამოცანების შესასრულებლად გვიწევს კონსოლიდან რაღაც ბრძანების შესრულება და რამდენიმე პარამეტრის გადაცემა. ამ პროცესის გასამარტივებლად შესაძლებელია package.json-ში scripts-ში დავამატოთ ეს ბრძანებები თავისი პარამეტრებით და მივანიჭოთ სახელი. შემდგომ ბრძანების გამოძახებისთვის საკმარისი იქნება მხოლოდ სახელის მითითება. მაგალითად თუ scripts-ში არის შემდეგი:

"scripts": {

"build": "browserify --entry index.js --outfile bundle.js",

},

browserify-ის ბილდის შესასრულებლად საკმარისი იქნება კონსოლში დავწეროთ:

>npm run build

1. **browserify**

npm მოდულების დაყენებისას მოდულები შესაძებელია იყოს სხვა და სხვა ფორმატში (commonjs, AMD ასევე არსებობს ES6 native). რადგან nodejs-გან გასხვავებით ბრაუზერებს ჯერ არ აქვთ მოდულების native მხარდაჭერა, საჭიროა პროექტის დაბილდვა, ანუ ყველა საჭირო მოდულის კოდის გაერთინაბა ერთ ან რამდენიმე ფაილში რომელსაც bundle ეწოდება. ამისთვის არსებობს სხვადასხვა ინსტრუმენტი მაგალითად browserify, webpack. Webpack-ს შედარებით მეტი შესაძლებლობა აქვს, თუმცა მისი კონფიგურირება შედარებით რთულია. ამიტომ თავიდან მაგალითებში გამოვიყენებთ browserify-ს.

Browserify-ის ბილდის გაშვების მაგალითი:

browserify --entry index.js --outfile bundle.js

პარამეტრები: entry - ჩვენი ძირითად js ფაილის სახელი, სადაც ხდება საჭირო მოდულების იმპორტი require-ს გამოყენებით. --outfile ბილდის შედეგად დასაგენერირებელი bundle ფაილის სახელი. სწორედ ამ დაგენერირებულ ფაილს ვამატებთ html-ში

1. **require, es6 import**

**require**

node-ს სტანდარტული api-ს გამოყენებით გარე მოდულის იმპორტისთვის გამოიყენება require (commonjs ფორმატში). მაგალითი:

const axios = require('axios');

**es6 import**

es6 სინტაქსით გარე მოდულის იმპორტი გამოიყურება შემდეგნაირად:

import axios from 'axios';

1. **babel, babelify**

**babel**

რადგან es6 სინტაქსის სრული მხარდაჭერა ბრაუზერებს ჯერ არ აქვთ, საჭიროა მისი transpiling-ი, ანუ ძველი სინტაქსით (es5) იგივე ფუნციონალის კოდის ავტომატური გენერირება. ამისთვის შექმნილია სხვადასხვა ინსტრუმენტი (transpiler), რომელთაგანაც ერთერთი ყველაზე გავრცელებული არის babel  
თითონ ბაბელის დასაყენებლად საჭიროა შემდეგი ბრძანების შესრულება:

>npm install --save-dev @babel/core

სხვადასხვა es6 feature-ს კომპილირებისთვის babel-ისთვის შექმნილია ცალკეული პლაგინები, მაგალითად @babel/plugin-transform-arrow-functions რომელიც => სინტაქსით დაწერილ ფუნქციებს გარდაქმნის სტადარტულ ფუნქციებად, რათა იმუშაოს ბრაუზერში რომელსაც არ აქვს ამის მხარდაჭერა. რადგან ცალკეული პლაგინების დაყენება საკმაოდ შრომატევადია, არსებობს პლაგინების მზა ნაკრებები ე. წ. preset. მაგალითად ყველა es2015+ ფუნქციონალის გამოსაყენებლად არსებობს მზა preset-ი env, რომლის დაყენება ცალკე არის საჭირო უშუალოდ ბაბელის დაყენების გარდა:

>npm install --save-dev @babel/preset-env

**babelify**

browserify ერთად გამოსაყენებლად შეიძლება babelify transform-ის დაყენება:

>npm install --save-dev babelify

scripts-ში შესაბამისად build ბრძანება შეიცვლება და იქნება შემდეგი:

browserify --entry index.js -o bundle.js -t [ babelify --presets [ @babel/preset-env] ]

1. **watchify**

კოდის ცვლილებისას ბილდის ყოველ ჯერზე ხელით გაშვების ასარიდებლად, არსებობს ინსტრუმენტები რომელიც ფაილებსი ცვლილებისას ავტომტურად უშვებენ ბილდს. ერთ ერთი ასეთია watchify, რომელიც ყველა იგივე პარამეტრს იღებს რაც browserify. მის დასაყენებლად საჭიროა შემდეგი ბრძანების შესრულება:

>npm install --save-dev babelify

და შემდეგ script-ში ჩამატება:

watchify --entry index.js -o bundle.js -t [ babelify --presets [ @babel/preset-env] ] -v

დავალება:

1. **L701.** შექმენით ცარიელი პროექტი და npm init-ის გამოყენებით შექმენით საწყისი package.json ფაილი
2. **L702** npm-ითდაამატეთ პროექტში browserify, (დაყენება + კონფიგი)
3. **L703** npm-ითდაამატეთ პროექტში babelify, babel, env preset-ით (დაყენება + კონფიგი)
4. **L704** npm-ითდააყენეთ პროექტში watchify კოდის ცვლილებისას ავტომატური ბილდისთვის
5. **L705** npm-ითდაამატეთ პროექტში jquery და მისი გამოყენებით (require) შეუცვლეთ გვერდს ფონი