**WEBPACK**

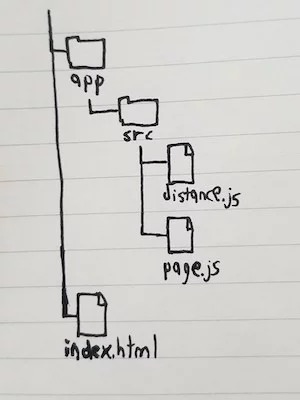
scr origin: <https://itnext.io/webpack-from-0-to-automated-testing-4634844d5c3c>

src translate: <https://tproger.ru/translations/webpack-from-zero-to-hero/>

Если не установлен yarn, то установщики здесь: <https://yarnpkg.com/ru/docs/install#windows-stable>

git init + создание удаленного репозитория (Это по желанию).

Создаем каталоги, общая структура каталогов такова:



Для работы с Webpack необходимо его установить с помощью npm или yarn:

yarn add --dev webpack webpack-cli

Теперь у нас подключён Webpack и доступна его командная строка.

Добавляем JS:

Основная структура приложения будет выглядеть так:

* У вас есть файл app/src/distance.js, экспортирующий функцию, которая запускает формулу вычисления расстояния (на самом деле нужно использовать [ортодому](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D1%82%D0%BE%D0%B4%D1%80%D0%BE%D0%BC%D0%B8%D1%8F)) и функцию, которая сообщает, какая точка из массива точек ближе всего находится.

// Функция distance() принимает 2 значения, представленных

// числами x и y, и возвращает расстояние между ними

//

// [TODO] Используйте ортодому

function distance(p2, p1) {

let yDist = p2.y - p1.y;

let xDist = p2.x - p1.x;

return Math.sqrt(Math.pow(yDist, 2) + Math.pow(xDist, 2));

}

// sortByDistance принимает ваше местоположение и массив точек

// и возвращает отсортированный массив точек

function sortByDistance(myPt, points) {

return points.sort(

(pt1, pt2) => distance(pt1, myPt) - distance(pt2, myPt));

}

module.exports = {distance, sortByDistance};

* Также у вас есть файл app/src/page.js, который использует код из distance.js, чтобы вывести ближайший магазин из списка, а затем отобразить его на странице.

import {sortByDistance} from './distance';

let stores = [

{name: "Cambridge Naturals", x: -71.1189, y: 42.3895},

{name: "Sarah's Market", x: -71.1311, y: 42.3823},

{name: "Whole Foods Fresh Pond", x: -71.1420, y: 42.3904},

];

let here = {name: "You are here", x: -71.1470, y: 42.3834};

let nearest = sortByDistance(here, stores)[0];

document.getElementById("nearest-store").innerHTML = nearest.name;

* И, наконец, у вас есть страница index.html.

<!DOCTYPE html>

<html>

<head>

<title>Ближайший магазин чая из гибискуса</title>

</head>

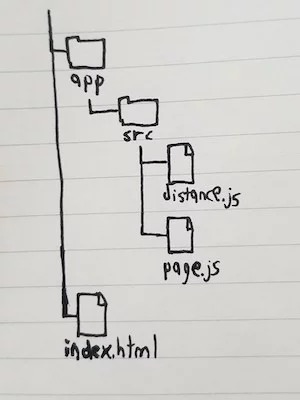
<body>

<p>Nearest store is <span id="nearest-store"></span></p>

<script src="dist/main.js"></script>

</body>

</html>



Отлично, теперь все зависимости JavaScript связаны. Используем Webpack для создания приложения. Выполним следующую команду:

**npx webpack app/src/page.js**

Это работает так: с помощью команды $ npx webpack app/src/page.js вы указываете, что отправной точкой (в терминологии Webpack — точкой входа в ваш код JavaScript) является page.js. Webpack читает файл page.js и видит в нём строку import {sortByDistance} from ./distance. Теперь он знает, что distance.js является зависимостью к page.js. И из всех зависимостей в вашем коде Webpack строит граф и использует его для построения пакетного JavaScript-файла dist/main.js

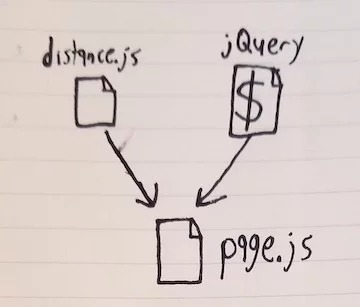
Установка jQuery:

**yarn add --dev jquery**

в page.js импортируем jquery:

**import $ from "jQuery";**

Теперь граф зависимостей выглядит так:



Для того чтобы нам постоянно не писать длинные команды по типу:

npx webpack app/src/page.js и т.д. в файл package.json необходимо добавить вот такие строки:

"scripts": {

"build": "webpack app/src/page.js"

}

Теперь, чтобы запустить сборку пакета, можно просто выполнить

**yarn build**

вместо $ npx webpack app/src/page.js.

**Настройка Webpack с помощью файла webpack.config.js**

Задать расположение точек входа и выхода программы можно, настроив файл конфигурации. Поместите следующий код в файл с именем webpack.config.js:

module.exports = {

entry: \_\_dirname + "/app/src/page.js",

output: {

path: \_\_dirname + "/dist/",

}

}

Теперь можно выполнить

**npx webpack**

или сделать ту же сборку (yarn build), что и раньше, без указания точки входа в программу в аргументах командной строки, т. к. теперь всё это есть в webpack.config.js. Также это значит, что нужно обновить скрипт файла package.json следующим образом:

"build": "webpack"

Файл конфигурации предназначен не только для настройки расположения входных и выходных файлов. Также можно настроить что именно Webpack делает, когда встречает файлы разных типов, используя специальные загрузчики, которые на самом деле являются JavaScript-программами, преобразующими ваш код.

**Загрузчики**

**Babel**

**yarn add --dev babel-core babel-loader@7.1.5 babel-preset-env**

Затем в файле webpack.config.js добавим следующий код в module.exports:

module: {

        rules: [

            {

                test: /\.js$/,

                exclude: ['/node\_modules/'],

                use: [

                    {

                        loader: 'babel-loader',

                        options: {

                            presets: ['env']

                        }

                    }

                ]

            }

        ]

    }

**Добавление тестовых сценариев в сборку**

Добавим несколько тестовых сценариев в файл distance.js. Для этого будем использовать Mocha и Chai. Выполните следующую команду:

**yarn add --dev mocha chai**

Затем создайте новый каталог app/test и новый файл app/test/distance.test.js, содержащий следующий фрагмент:

describe("distance", function() {

it("calculates distance with the good ol' Pythagorean Theorem", function() {

let origin = {x: 0.0, y: 0.0};

let point = {x: 3.0, y: 4.0};

expect(distance(point, origin)).to.equal(5);

});

});

describe("sortByDistance", function() {

it("sortsByDistance", function() {

let places = [

{name: "Far away", x: 100, y: 50},

{name: "Nearby", x: 20, y: 10},

];

let origin = {name: "Origin", x: 0, y: 0};

let sorted = sortByDistance(origin, places);

expect(sorted[0].name).to.equal("Nearby");

expect(sorted[1].name).to.equal("Far away");

});

});

Так как слово import, Node в данный момент не поддерживает, поэтому чтобы distance.test.js увидел функции distance и sortByDistance их необходимо подключать при помощи require():

const chai = require('chai'); // require chai

const expect = chai.expect;

const distancefunc = require('../src/distance');

const distance = distancefunc.distance;

const sortByDistance = distancefunc.sortByDistance;

В свою очередь distance и sortByDistance должны быть экспортированны из своего файла.

module.exports = {distance, sortByDistance};

Далее допишем webpack.config.js:

let glob = require("glob");

let entry = \_\_dirname + "/app/src/page.js";

let outputPath = \_\_dirname + "/dist/";

if (process.env.TESTBUILD) {

 entry = glob.sync(\_\_dirname + "/app/test/\*\*/\*.test.js");

 outputPath = \_\_dirname + "/test-dist/";

}

module.exports = {

    entry: entry,

    output: {

        path: outputPath

    },

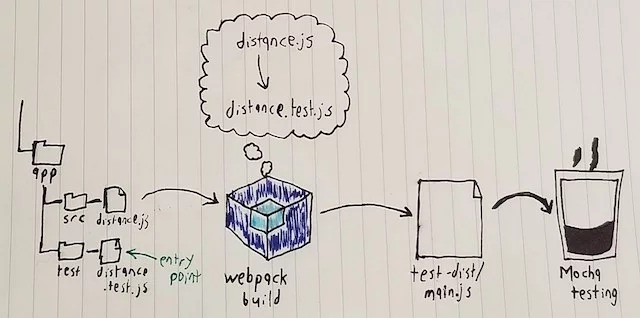
. . . . . .  
. . . . . .

. . . . . .

Список тестовых инструкций:

1. Webpack строит граф зависимостей, начинающийся с тестовых файлов, а не с файлов приложения.
2. Webpack создаёт файл JavaScript, содержащий весь тестовый код и его зависимости без ключевого слова import.
3. Выполняются тесты, запуская Mocha для этого JavaScript-файла.

Всё это будет выглядеть следующим образом:



Если выполнить **TESTBUILD=true npx webpack**, то можно увидеть каталог test-dist.

А если запустить **npx mocha test-dist/main.js**, то можно увидеть, как выполняются наши тесты.

Объединим эти вызовы в один, для этого создадим alias в package.json чтобы не писать длинные команды в консоль:

"test": "TESTBUILD=true npx webpack && mocha test-dist/main.js && npx rm -rf test-dist"

на **WIN** нужен бубен:  
необходимо установить process-env.  
yarn add -D cross-env  
Затем в разделе scripts немного изменил вызов команд:  
"test": "cross-env TESTBUILD=true npx webpack && mocha test-dist/main.js && npx rm -rf test-dist"  
источник: <https://medium.com/@hatzlgerhard/great-article-super-explained-863612b630fb>

теперь мы можно просто набрать:

**yarn test**

**Маппинг исходных файлов тестового кода**

Чтобы Mocha мог использовать source map при запуске тестов, необходимо установить ещё один пакет:

**yarn add --dev source-map-support**

Затем обновим webpack.config.js:

let glob = require("glob");

let entry = \_\_dirname + "/app/src/page.js";

let outputPath = \_\_dirname + "/dist/";

let devtool = "";

if (process.env.TESTBUILD) {

    entry = glob.sync(\_\_dirname + "/app/test/\*\*/\*.test.js");

    outputPath = \_\_dirname + "/test-dist/";

    devtool = "source-map";

}

Затем в объект module.exports добавим строку:   
devtool: devtool

Затем нужно обновить скрипт Mocha в разделе scripts.test файла package.json:

test": "cross-env TESTBUILD=true npx webpack && mocha test-dist/main.js --require source-map-support/register && rmdir /s/q test-dist"

Теперь, если вы выполните $ yarn test и получите ошибку, вы увидите, в какой строке она находится.

ergerg