hillel IT school

Lesson 27

План заняття

- package.json;
- NPM;
- Gulp;

NPM

npm (Node Package Manager) – дефолтний пакетний менеджер JavaScript, що працює на Node.js. Менеджер npm складається із двох частин:

CLI (інтерфейс командного рядка) – засіб для розміщення та завантаження пакетів, онлайн-репозиторії, що містять JS пакети.

Структуру репозиторію npmjs.com можна уявити, як центр виконання замовлень, який отримує товари (npm-пакети) від продавців (автори пакетів) та розповсюджує ці товари серед покупців (користувачі пакетів).

Пакети можна знайти тут https://www.npmjs.com/

Документація тут https://docs.npmjs.com/

Важливо!

В жодному разі не пушимо у гіт node_modules

Для того щоб кожен розробник мав змогу користуватись вашим проєктом існує package.json

Він зберігає список пакетів, необхідних для проекту із потрібними версіями, і на іншій машині ми можемо легко встановити всі пакети

Базові команди

npm init - Ставить вам різні питання та створює package.json, який визначає налаштування проекту, залежності, скрипти, назву та інше. Флаг --yes встановить на всі запитання дефолтні відповіді.

npm install - Встановить всі пакети та залежності.

npm install | i packageName - встановить пакет та додасть його до package json

npm start - Виконає скріпт start

Npm test - Виконає скріпт test

npm run start:dev - Виконає кастомний скріпт start:dev

npm uninstall packageName - Видалить пакет packageName

npm update - Зробить апдейт паакетів

Усі команди npm тут https://docs.npmjs.com/cli/v6/commands

package.json

package.json - Це JSON файл, який живе в керуючому центрі вашого проекту. Він містить human-readable metadata про проект (name і description), ваші пакети, номера версії, та скріпти.

Документція тут https://docs.npmjs.com/cli/v10/configuring-npm/package-json

main - визначає точку входу у ваш проект і зазвичай файл, який використовується для запуску проекту. Зазвичай це файл index.js у кореневій папці вашого проекту, але це може бути будь-який файл, який ви вирішите використовувати як основну точку входу до вашого пакету.

scripts - приймає об’єкт із його ключами, які є сценаріями, які ми можемо запускати за допомогою npm run <scriptName>, а значенням є фактична команда, яка виконується.

package.json

dependecies - Усі залежності, які використовує ваш проект (зовнішній код, на який покладається проект), перераховані тут. Коли пакет інсталюється за допомогою npm CLI, він завантажується у вашу папку node_modules/ і додається запис до вашої властивості залежностей із зазначенням назви пакета та встановленої версії. Це об'єкт із назвами пакетів як ключами та версією або діапазоном версій як значенням. З цього списку npm знає, які пакунки отримати та встановити (і в яких версіях), коли npm install запускається в каталозі. Поле залежностей вашого package.json є основою вашого проекту та визначає зовнішні пакети, які потрібні вашому проекту.

Каретки (^) і тильди (~), які ви бачите у версіях залежностей, є позначеннями для діапазонів версій, визначених у SemVer.

devDependencies - Подібно до поля залежностей, але для пакунків, які потрібні лише під час розробки, і не потрібні у виробництві.

Прикладом може бути використання інструменту для перезавантаження вашого проекту під час розробки, як-от nodemon, який ми не використовуємо після розгортання програми та її виробництва. Запис devDependency — чудовий спосіб задокументувати, які інструменти потрібні для програми під час розробки. Щоб установити пакет із npm як devDependency, ви можете запустити npm install --save-dev <package>.

Системи збирання

Основні задачі:

1. Компіляція CSS та розуміння препроцесорів.
2. Мініфікація та конвертація JS та CSS.
3. Автоматичне конвертування шрифтів та їх підключення.
4. ES6 модулі та транспайлінг.
5. Опрацювання зображень.
6. Відстеження змін у файлів та перезаванаження браузера.
7. Створення дев сервера.
8. Розбиття файлів на chsnks.
9. Усі інші кайфушки....

Готові рішення

Ви можете користуватись готовими системами збирання.

Наприклад Prepross https://prepros.io/ але це не наші методи. Ми тру кодери...

Все почалося з....

Grunt - це таск ранер https://gruntjs.com/

Одним словом: автоматизація. Чим менше роботи вам доведеться робити під час виконання повторюваних завдань, таких як мініфікація, компіляція, модульне тестування, лінтування тощо, тим легшою стане ваша робота. Після того, як ви налаштували його за допомогою Gruntfile, програма для виконання завдань може виконувати більшість цієї повсякденної роботи за вас — і вашу команду — практично без зусиль.

Чому ми його не використовуємо:

1. Він старий.
2. Зоснован на файлах а не на потоках. Щоб запустити наступну задачу, попередня мусить зберегтись у файлі.
3. Дуже не зручний процес конфігурування.

Gulp

Gulp - простий таск ранер який замінив Grunt.

Його переваги:

1. Простота конфігурації.
2. Конфігурується за домогою імперативного підхіду (опис у js файлі використовуючи js). Запускається у консолі.
3. Працює за допомогою потоків.
4. Зручний для багатьох проектів (навідь для Wordpess apps).

Сайт - https://gulpjs.com/

Гітхаб - https://github.com/gulpjs/gulp

Плагіни - https://gulpjs.com/plugins

Чому ми говорим про Gulp? Це просте рішення для простих проектів, та його ліпше зрозуміти на старті.

Gulp

Gulp - простий таск ранер який замінив Grunt.

Його переваги:

1. Простота конфігурації.
2. Конфігурується за домогою імперативного підхіду (опис у js файлі використовуючи js). Запускається у консолі.
3. Працює за допомогою потоків.
4. Зручний для багатьох проектів (навідь для Wordpess apps).

Сайт - https://gulpjs.com/

Гітхаб - https://github.com/gulpjs/gulp

Плагіни - https://gulpjs.com/plugins

Чому ми говорим про Gulp? Це просте рішення для простих проектів, та його ліпше зрозуміти на старті.

Gulp

Ідея...

Описуємо що нам потрібно робити, запускаєто Gulp, він робить усе інше. Ми просто кодимо.

Наприклад

Мені потрібно запустити відслідкування файлів js, кожен раз як я буду вносити зміни, перезавантажуй мені сторінку у браузері. Коли я закінчу роботу, зроби мені фінальні, зібрані файли (bundle) та задеплой йох на хостінг.

Далі використовуючи декілька команд ми кажемо Gulp що і коли робити.

Gulpfile

Gulpfile — це файл у каталозі вашого проекту під назвою gulpfile.js, який автоматично завантажується під час виконання команди gulp. У цьому файлі ви часто побачите API gulp, наприклад src(), dest(), series() або parallel(), але можна використовувати будь-які модулі JavaScript або Node. Будь-які експортовані функції будуть зареєстровані в системі завдань gulp.

Tasks

Кожне завдання gulp є асинхронною функцією JavaScript — функцією, яка приймає зворотний виклик спочатку помилки або повертає потік, обіцянку, джерело події, дочірній процес або спостережуваний. Через деякі обмеження платформи синхронні завдання не підтримуються, хоча є альтернатива.

Завдання можна вважати публічними або приватними.

- Загальнодоступні завдання експортуються з вашого gulpfile, що дозволяє запускати їх за допомогою команди gulp.
- Приватні завдання призначені для внутрішнього використання, зазвичай використовуються як частина послідовної () або паралельної () композиції.

Наприклад:

default - загальне завдання

sayHello - приватне завдання

Compose tasks

Gulp надає два потужні методи композиції, series() і parallel(), що дозволяє компонувати окремі завдання у більші операції. Обидва методи приймають будь-яку кількість функцій завдання або складених операцій. series() і parallel() можуть бути вкладені в себе або один в одного на будь-яку глибину.

Щоб ваші завдання виконувалися в порядку, використовуйте метод series().

Щоб завдання запускалися з максимальною паралельністю, поєднайте їх із методом parallel().

series() і parallel() можуть бути вкладені на будь-яку довільну глибину.

Working with Files

Методи src() і dest() надаються gulp для взаємодії з файлами на вашому комп’ютері.

src() отримує glob для читання з файлової системи та створює потік Node. Він знаходить усі відповідні файли та зчитує їх у пам’ять для проходження через потік.

Потік, створений src(), має повертатися із завдання, щоб сигналізувати про асинхронне завершення, як зазначено у створення завдань.

Основним API потоку є метод .pipe() для з’єднання потоків Transform або Writable.

Working with Files

Методи src() і dest() надаються gulp для взаємодії з файлами на вашому комп'ютері.

src() отримує glob для читання з файлової системи та створює потік Node. Він знаходить усі відповідні файли та зчитує їх у пам'ять для проходження через потік.

Потік, створений src(), має повертатися із завдання, щоб сигналізувати про асинхронне завершення, як зазначено у створення завдань.

Основним API потоку є метод .pipe() для з'єднання потоків Transform або Writable.

dest() отримує вихідний рядок каталогу, а також створює потік Node, який зазвичай використовується як потік-завершення. Коли він отримує файл, який передається через конвеєр, він записує вміст та інші деталі у файлову систему в заданому каталозі. Метод symlink() також доступний і працює як dest(), але створює посилання замість файлів (див. symlink() для отримання додаткової інформації).

Найчастіше плагіни розміщуються між src() і dest() за допомогою методу .pipe() і перетворюють файли в потоці.

src()

Створює потік для читання об'єктів із файлової системи.

Синтаксис:

src(globs, [options])

- globs - масив або рядок шляхів для перегляду у файловій системі.
- options - об'єкт параметрів.

dest()

Створює потік для запису об’єктів у файлову систему.

Синтаксис:

dest(directory, [options])

- directory - строка або функція вказує шлях вихідного каталогу, куди будуть записані файли.
- options - об’єкт параметрів

pipe()

Pipe - це канал, який зв'язує потік для чтиння і потік для запису і дозволяє відразу зчитати з потоку чтиння в потік запису.

Детальніше розглянемо у NodeJS

pipe()

Pipe - це канал, який зв'язує потік для чтиння і потік для запису і дозволяє відразу зчитати з потоку чтиння в потік запису.

Детальніше розглянемо у NodeJS

Globs

Glob — це рядок літералів та/або символів підстановки, які використовуються для відповідності шляхів до файлів. Globbing — це акт пошуку файлів у файловій системі за допомогою одного або кількох globs.

- * (single-star) - Збігається з будь-якою кількістю символів в одному сегменті (включно з жодним). Корисно для об'єднання файлів в одному каталозі.
- ** (double-star) - Збігається з будь-якою кількістю символів у сегментах, навіть жодного. Корисно для глоббування файлів у вкладених каталогах.
- ! (negative) - Коли ми використовуємо масив з globs негативний glob повинен слідувати принаймні за одним невід’ємним glob у масиві. Перший знаходить набір збігів, потім негативний видаляє частину цих результатів. Виключаючи всі файли в каталозі, ви повинні додати /** після назви каталогу, яку бібліотека globbing оптимізує внутрішньо.

Using Plugins

Плагіни Gulp — це потоки перетворення вузлів, які інкапсулюють загальну поведінку для перетворення файлів у конвеєрі — часто розміщуються між src() і dest() за допомогою методу .pipe(). Вони можуть змінювати назву файлу, метадані або вміст кожного файлу, який проходить через потік.

Плагіни від npm за допомогою ключових слів «gulpplugin» і «gulpfriendly» можна переглядати та шукати на сторінці пошуку плагінів.

Кожен плагін повинен виконувати лише невелику кількість роботи, тому ви можете з'єднувати їх як будівельні блоки. Щоб отримати бажаний результат, вам може знадобитися поєднати їх кілька.

Ви можете підключати плагіни за умовами, використовуючи плагін gulp-if https://www.npmjs.com/package/gulp-if

Watching Files

API watch() підключає globs до завдань за допомогою спостерігача файлової системи. Він стежить за змінами у файлах, які відповідають glob, і виконує завдання, коли відбувається зміна. Якщо завдання не сигналізує про асинхронне завершення, воно ніколи не буде запущено вдруге.

Завдання спостерігача не може бути синхронним, як завдання, зареєстровані в системі завдань. Якщо ви передаєте завдання синхронізації, завершення не можна визначити, і завдання не буде запущено знову – передбачається, що воно все ще виконується.

За замовчуванням спостерігач виконує завдання кожного разу, коли файл створюється, змінюється або видаляється. Якщо вам потрібно використовувати різні події, ви можете скористатися опцією подій під час виклику watch(). Доступні події: «add», «addDir», «change», «unlink», «unlinkDir», «ready», «error». Крім того, доступний «all», який представляє всі події, крім «ready» і «error».

Ви можете зробити так щоб завдання чекали першої зміни у файлі, за допомогою ignoreInitial: false

Також можете зробити затримку перед виконанням завдання, за допомогою delay: 500

SCSS

Створимо задачу на те щоб компілювати SCSS у CSS

Для цього підключимо препроцесор sass https://www.npmjs.com/package/sass

Репозіторій https://github.com/sass/dart-sass

Сайт https://sass-lang.com/

Та дя того щоб gulp розумів що робити з цими файлами gulp-sass https://www.npmjs.com/package/gulp-sass

Concat

Створимо задачу на те щоб об’єднати усі js файли до одного результуючого.

Для цього використовуємо gulp-concat https://www.npmjs.com/package/gulp-concat

Babel

Створимо задачу на те щоб можна було використовувати сучасний js синтаксис та його розуміли усі браузери.

Для цього використовуємо gulp-babel https://www.npmjs.com/package/gulp-babel

Гітхаб https://github.com/babel/gulp-babel

Що таке babel?

Babel — це ланцюжок інструментів, який в основному використовується для перетворення коду ECMAScript 2015+ у зворотно сумісну версію JavaScript у поточних і старих браузерах або середовищах. Ось основні речі, які Babel може зробити для вас:

- Перетворення синтаксису
- Функції Polyfill, яких немає у вашому цільовому середовищі (через стороннє polyfill, наприклад core-js)
- Перетворення вихідного коду (codemods)
- І більше...

Детальніше тут https://babeljs.io/

Html

Створимо задачу на те щоб можна було розбивати html на темплейти та створювати один результуючий html.

Для цього використовуємо gulp-ssi https://www.npmjs.com/package/gulp-ssi

Гітхаб https://github.com/etylsarin/gulp-ssi

Що таке ssi (Server Side Includes for NodeJS)?

Це бібліотека яка надає нам змогу вставляти частини коду у різні файли.

Гытхаб https://github.com/kidwm/node-ssi

Дякую за увагу