МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Київський національний університет будівництва і архітектури

КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Конспект лекцій

для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальностями
122 **“Комп’ютерні науки”,**
126 **“Інформаційні системи і технології”**

Київ 2024

УДК 681.3
К83

Укладачі: Т.А. Гончаренко, канд. техн. наук, доцент;
В.М. Хроленко, канд. техн. наук, доцент;
В.Г. Голенков, старший викладач

Рецензент О.О. Терентьєв, д-р техн. наук, професор

Відповідальна за випуск Т.А. Гончаренко, канд. техн. наук, доцент

Затверджено на засіданні кафедри інформаційних технологій, протокол № 10 від 18 квітня 2024 року

В авторській редакції

Крос-платформне програмування: конспект лекцій. /Уклад.: К83 Т.А. Гончаренко, В.М. Хроленко, В.Г. Голенков. – Київ: КНУБА, 2024. – 54 с.

Містить теоретичні відомості з дисципліни “Крос-платформне програмування”, які сприяють закріпленню та поглибленню здобувачами теоретичних знань з дисципліни та дозволяють здобувачам набути практичних навичок роботи з платформою для розробки веб-сайтів Node JS.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальностями 122 “Комп’ютерні науки”, 126 “Інформаційні системи і технології”

Зміст

Лекція 1. ВСТУП ДО NODE JS І EXPRESS

За декілька останніх років Node JS став неймовірно популярною платформою для розробки різних веб сайтів. В ході курсу навчимося працювати з Node, ознайомимося з його синтаксисом, створимо кілька простих веб додатків, а також розглянемо фреймворк Express, який був розроблений спеціально під Node JS і є відмінним доповненням. Перед початком перегляду курсу варто вивчити HTML, а також JavaScript. HTML необхідний для написання розмітки на сайті, а вивчивши додатково JS вам буде набагато простіше розуміти що відбувається в курсі, так як Node JS всюди використовує синтаксис мови JavaScript.

Node JS здобув велику популярність завдяки можливості писати на єдиній мові локальні завдання і задачі пов'язані з серверної роботою і базами даних. Платформа побудована на движку V8 з використанням мов JavaScript, C і C++. У свою чергу, V8 - це движок JS, який має відкритий програмний код. Платформа відмінно підходить для написання серверних додатків і чатів, але також можна писати десктопні програми за допомогою NW JS або ж AppJS. Є можливість легкої інтеграції різного API (інтерфейс програмування застосунків), що значно прискорює загальний час розробки програм.

Node JS також має свій пакетний менеджер, який називається npm. Завдяки йому ви можете підключати сторонні бібліотеки, запускати локальний сервер і робити ще безліч цікавих речей.

На сьогоднішній день Node може похвалитися великим списком веб сайтів, що розроблені з використанням даної технології. Список декількох відомих веб-сайтів на Node JS:

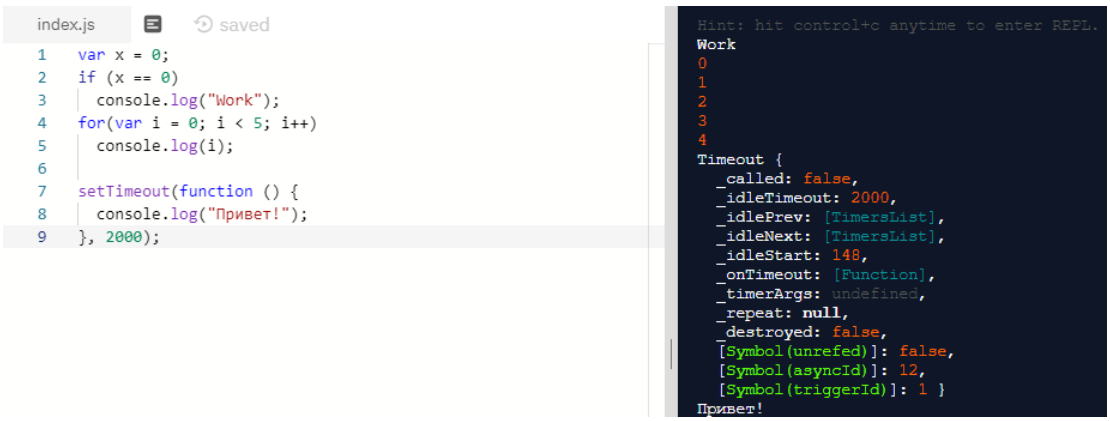
- Всесвітня платіжна система - PayPal;
- Одна з популярних пошукових систем світу - Yahoo;
- Онлайн версія газети Wall Street Journal;
- Мережа для збору кращих фотографій та відео з соц. мереж - Storify.
- Сайт, який працює з пацієнтам і кращими місцевими постачальниками медичних послуг - Opencare.

Движок V8 та глобальні об’єкти

Движок V8 з'явився ще в далекому 2008 році. Його розробкою займалося один з підрозділів компанії Google. На сьогоднішній момент це потужна технологія, написана на основі мов C++ і JavaScript. Мета движка полягає в конвертації JavaScript коду в C++, а потім в машинний код. Таким чином розробники можуть писати звичний синтаксис на мові JavaScript, а движок робить всю «брудну» роботу за них, перетворюючи код в більш складний C++ і далі в абсолютно незрозумілий для людини машинний код.

Платформа Node JS була написана на C++, тому в неї було нескладно впровадити движок V8. Node.js пропонує спеціальний об'єкт global, який надає доступ до глобальних, тобто доступних з кожного модуля програми, змінних і функцій. Зразковим аналогом даного об'єкта в JavaScript для браузера є об'єкт window. Однак по можливості рекомендується уникати визначення і використання глобальних змінних, і переважно орієнтуватися на створення змінних, інкапсульованих в рамках окремих модулів.

В Node можна спокійно прописати всі стандартні структури з JavaScript: цикли, умови, змінні, функції і багато іншого. Для наочності продемонструємо цю можливість на простому прикладі:



Таким чином, ми бачимо, що код, написаний мовою JavaScript, легко працює з платформою Node JS, і під час компіляції не виникає синтаксичних помилок.

Контрольні запитання

1. Яке ризначення платформи Node JS?
2. Яке призначення движка V8?

Лекція 2. ФУНКЦІЇ ТА МОДУЛІ NODE JS

Функції, модулі та директива require()

Кожна програма на Node JS складається з певної кількості модулів, які підключаються за допомогою директиви require. В даній темі ми розглянемо і навчимося працювати з цими засобами.

Якщо ви створюєте велику або навіть середню програму на Node JS, то ви завжди будете ділити її на різні модулі. Кожен модуль являє собою окремий файл, який наповнений різним функціоналом. Ви можете підключати модулі до інших файлів і використовувати функції, прописані в модулях.

Таким чином сайт буде добре структурований, а код буде більш читабельним. Для створення модуля необхідно створити новий файл, в якому прописуються іменовані функції, змінні та інші конструкції:

```
var some = function() {
    console.log("Елементарна функція");
};
```

В наведеному вище прикладі створена іменна функція. Для використання функції поза модулем її необхідно експортувати. Приклад експорту з модуля:

```
module.exports = some;
```

Для підключення модуля в інших файлах необхідно використовувати директиву require () і в ній прописати повний шлях до файлу:

```
var some = require('./file'); // file - назва під'єднаного файлу
```

Тепер можна використовувати функцію *some* всередині файлу з підключеним модулем. Розглянемо більш детально практичне використання модулів та функцій, наприклад, щоб дізнатись довжину заданого числового масиву *array*.

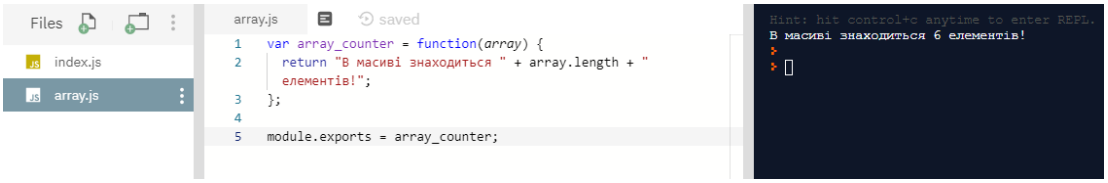
Створюємо файл *array.js*, де прописуємо функцію *array_counter*, яка і повертатиме кількість елементів масиву.

```
var array_counter = function(array) {
  return "В масиві знаходиться " + array.length + " елементів!";
};

module.exports = array_counter;
```

Разом з цим вказуємо метод *exports*, що дозволить нам експортувати функцію з даного модуля. Після цього створюємо новий модуль, де задаємо масив чисел.

```
var counter = require('./array');

console.log(counter([1, 7, 99, 8, 45, 8]));
```

Функція Console.log дозволяє вивести результат на екран, який представлено нижче:



Множинне виведення з модуля

В попередній темі вже розглянули роботу з модулями, тож тепер навчимося експортувати кілька функцій або методів з одного модуля. Зовсім незручно створювати модуль для роботи лише з одним методом. Саме тому в Node JS існує множинний експорт даних з модуля. Для реалізації такого існує три способи.

По-перше, можна використовувати конкретні властивості для виведення:

```
module.exports.add = add;
```

Тут вказується ім'я властивості для експорту, а також відбувається присвоювання до нього значення. Значенням може бути змінна або ціла функція. Подібних властивостей можна експортувати незліченну кількість.

По-друге, можна експортувати значення без створення змінних:

```
module.exports.some_value = "Експорт рядка";
```

Так само можна присвоювати іменовані функції для експорту без створення змінних.

По-третє, можна експортувати відразу цілий масив властивостей і значень:

```
module.exports = {
    variable: 23.5,
    adding: adding
};
```

Таким чином ми експортуємо відразу множину функцій і змінних. Щоб скористатися експортованими даними необхідно імпортувати модуль в інший файл і створити змінну модуля. Далі ви можете через крапку звертатися до всіх експортованих даних:

```
var our_module = require('./src_file'); // Імпорт модуля
console.log(our_module.val); // Використання даних
```

Завдяки такому запису ми з легкістю можемо прописувати в модулі всі необхідні функції, після чого експортувати їх і викликати в будь-яких інших файлах. Для прикладу створимо новий файл things.js, де використаємо відразу декілька функцій – виведення, підрахунку елементів в масиві та множення.

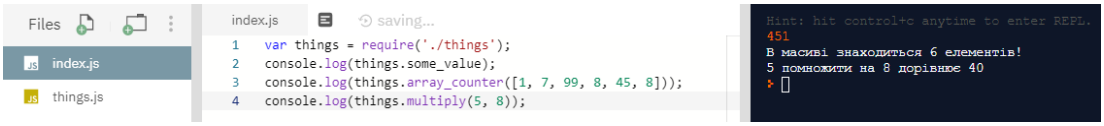
Додаємо до попереднього прикладу дві нові функції:

```
var multiply = function(x, y) {
  return `${x}помножити ${y}дорівнює ${x * y}`;
};

var some_value = 451;
```

Для експортування даних записуємо відповідний метод:

```
module.exports = {
  array_counter: array_counter,
  multiply: multiply,
  some_value: some_value
};
```

Далі створюємо новий файл index.js для підключення та виконання функцій і виведення результату на екран:



Відстеження подій у Node

Щоб виконувати будь-які дії після натиснення на кнопку або при наведенні мишки необхідно відстежувати події і виконувати код. В даній темі розглянемо відстеження подій в Node JS.

Node JS володіє потужним модулем, здатним обробляти події, а також викликати події вручну. За рахунок модуля *events* ви можете використовувати метод EventEmitter для відстеження подій.

Структура створення події дуже схожа на синтаксис jQuery. Спершу необхідно створити назву для події або ж використовувати стандартні назви. Далі необхідно прописати функцію для реалізації самої події. Приклад створення події:

```
var em = new events.EventEmitter(); // Обробник подій
em.on('event_name', function() { // Створення події
    // Вивід інформації при спрацьовуванні події
    console.log("Ця подія спрацювала!");
});
```

Для виклику подібних подій можна скористатися функцією *emit* і в ній вказати ім'я необхідної події. Також можна використовувати модуль *util*, який дозволяє додати наслідування даних для об'єктів. Щоб створити успадкування необхідно використовувати змінну модуля *util* і функцію *inherits*.

Розглянемо більш детально функцію *Emit* на прикладі:

```
var events = require('events');

var myEmit = new events.EventEmitter();

myEmit.on('some_event', function(text) {
  console.log(text);
});

myEmit.emit('some_event', 'Обробник подій спрацював!');
```

Спочатку викликаємо вбудований модуль *events*, за допомогою якого створюємо подію – myEmit, яка при спрацьовуванні буде виконувати функцію виведення рядка тексту.

Розглянемо роботу з модулем *Util* на прикладі:

```
var events = require('events');
var util = require('util');

var Cars = function(model) {
  this.model = model;
};

util.inherits(Cars, events.EventEmitter);

var bmw = new Cars('BMW');
var audi = new Cars('Audi');
var volvo = new Cars('Volvo');

var cars = [bmw, audi, volvo];
cars.forEach(function(car) {
  car.on('speed', function(text) {
    console.log (car.model + " speed is - " + text);
  });
});
```

Створюємо змінну, якій присвоюємо властивості модуля *util*, а далі описуємо конструктор – Cars. За допомогою функції *inherits* ми дозволяємо наслідування обробника подій всім об'єктам створеним за допомогою конструктора. А далі створюємо об'єкти конструктора, і за допомогою циклу forEach виводимо інформацію про модель автомобіля, використовуючи раніше створену функцію. Отже, модуль util дозволяє нам спростити запис обробника подій за допомогою наслідування.

Написання та читання файлів

Node JS має безліч модулів, які дозволяють робити приголомшливі речі. В даній темі розглянемо модуль *fs*, який додає можливість читання і запису даних в файли.

У Node JS існує модуль з назвою *fs* (file system), що відповідає за роботу з файлами. Завдяки цьому модулю можна працювати з файлами (створювати, видаляти, записувати і зчитувати дані), директоріями, а також з потоками даних. Про додаткові можливості розглянемо в наступних темах, а в цій лише торкнемося теми читання і запису файлів як в синхронному, так і в асинхронному режимі. Спочатку для роботи з файлами необхідно підключити модуль:

```
var fs = require('fs');
```

Заведено називати змінні модуля так само як і сам модуль. Щоб зчитувати дані використовуйте метод fs.readFileSync (), а щоб записати - fs.writeFileSync (). Завжди краще використовувати не синхронну роботу з файлами, а асинхронну. Для асинхронної роботи використовуйте ті ж самі

методи, але без слова «Sync». Також вказуйте в якості останнього параметра функцію, що спрацює по завершенню роботи з файлом.

Розглянемо приклад синхронної роботи з файлом:

```
var fs = require('fs');

var file_readed = fs.readFileSync('text.txt', 'utf8');
var message = "Привіт світ!\n" + file_readed;
fs.writeFileSync('some_new_file.txt', message);
```

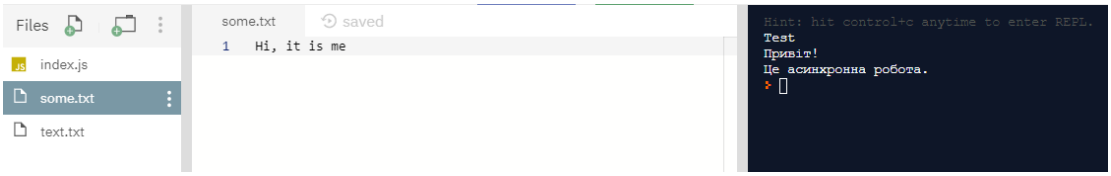
Синхронна робота означає те, що код, який йде після неї не буде виконана, доки не завершиться зчитування або запис в файл. Для початку роботи ми підключаємо модуль *fs*. Далі в методі *fs.readFileSync* в дужках вказуємо спочатку файл для читання, шлях до нього, якщо він знаходиться в іншій папці, а також зазначаємо тип кодування символів. Потім для запису використовуємо метод *fs.writeFileSync*, в якому вказуємо назву файлу та текст, який можна записати у змінну або задати явно.

Розглянемо приклад асинхронної роботи:

Принципова різниця з синхронною роботою в тому, що читання і запис в файл може відбуватись паралельно з іншими частинами коду, що значно прискорює процес виконання. Розглянемо даний приклад детальніше:

```
var fs = require('fs');

fs.readFile('text.txt', 'utf8', function(err, data) {
  console.log(data);
});

fs.writeFile('some.txt', 'Hi, it is me', function(err, data) {});

console.log("Test");
```

Спочатку ми так само як і в минулому прикладі зчитуємо дані з файлу text.txt. Але тепер з’являється нова функція в методі, яка необхідна для того, щоб обробити виключення або при вдалій спробі зчитати дані з файлу, і потім вивести їх в консоль. За таким принципом відбувається і запис тексту в файл some.txt. У результаті виконання всього коду спочатку буде виведено слово Test, оскільки процедура виводу тексту на екран значно швидша за зчитування та запис в файл.



Робота з директоріями

У минулій темі розглянуто роботу з файлами і тепер настав час навчиться працювати з директоріями. Ми навчимося створювати і видаляти директорії за допомогою Node JS.

У Node JS є модуль з назвою fs, що відповідає за роботу з файлами і директоріями. Ми його вже розглядали, але тепер розглянемо його можливості щодо роботи з папками. Спочатку для роботи з файлами або ж папками необхідно підключити модуль:

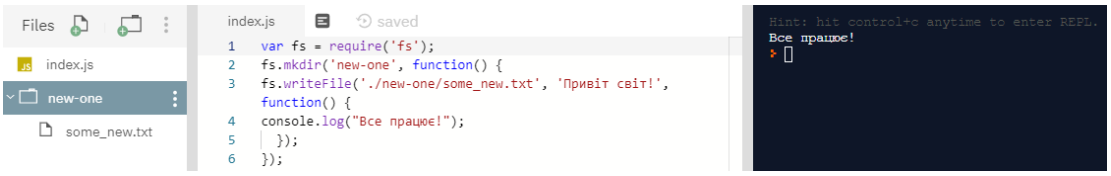
```
var fs = require('fs');
```

Для створення папки необхідно використовувати метод *mkdir*(), а для видалення *rmdir*(). В якості першого параметра необхідно передавати назву файлу, а в якості другого параметра функцію, що спрацює по завершенню роботи команди.

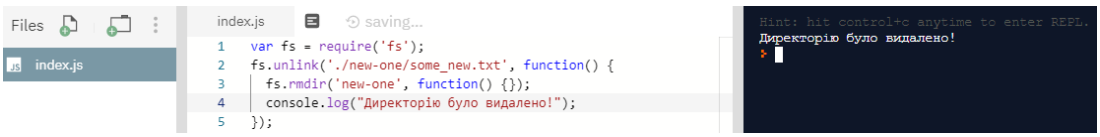
Якщо ви видаляєте цілу папку, то переконаєтеся що в ній немає ніяких додаткових файлів або ж папок. Якщо такі є, то спочатку необхідно видалити файли, а потім папку. Для видалення файлів різного типу використовуйте метод *unlink* ():

```
fs.unlink('./some_image.png', function() {});
```

Приклад реалізації:



Даний код створює папку з текстовим файлом some_new.txt, у який записується текст. Тепер спробуємо видалити директорію:



Як ви бачите, код спрацював і директорії більше немає в списку зліва.

Робота та створення сервера на Node

Node JS славиться тим, що ви можете писати на JS, але при цьому працювати з сервером. В даній тему ми підтвердимо цю теорію і створимо свій локальний сервер на якому виведемо сторінку з текстом.

Завдяки модулю *http* можна створювати і підключатися до сервера буквально за пару рядків коду. В ході теми ми створили локальний сервер, на якому вивели текст без використання HTML-тегів.

Для створення сервера необхідно підключитися до модуля *http* і використовувати метод createServer. У метод необхідно помістити функцію з двома параметрами: *request* (запит) і *response* (відповідь).

У параметрі response ми можемо встановити яку версію має повернути сервер, а також що має бути відображено на сторінці сайту. У заголовках ми вказуємо типи даних, які ми хочемо передати і відобразити на сайті.

В даний момент ми працюємо з простим типом даних, а саме зі звичайним текстом - text / plain. При роботі з заголовками важливо вказати кодування, щоб текст написаний на кирилиці виводився коректно.

Для виведення інформації в браузері використовуйте метод end:

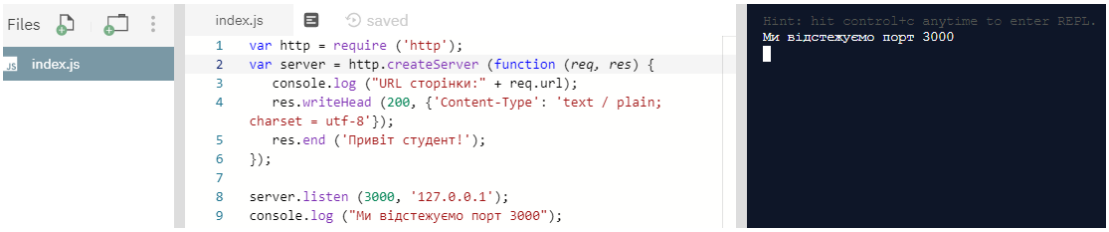
```
var http = require('http'); // Необхідний модуль

// Створення сервера
var server = http.createServer(function(req, res) {
    // Зазначення заголовків (тип даних і кодування)
    res.writeHead(200, {'Content-Type': 'text/plain; charset=utf-8'});
    // Текст, який буде відображатись на сторінці
    res.end('Просто звичайний текст без HTML');
});
```

Також необхідно встановити саме з'єднання, в якому вказати порт, а також адреса сервера. У нашому випадку ми використовуємо локальні параметри:

```
// server - змінна, що створена раніше
server.listen(3000, '127.0.0.1');
```

Якщо запустити програму і зайти на адресу 127.0.0.1:3000, то на сайті буде напис: «Просто звичайний текст без HTML».

Ще один приклад створення локального сервера:



Після виконання коду було створено сервер, при переході за відповідною адресою на екрані з’явиться текст «Привіт студент!».

Робота з потоками у Node JS

В цій темі ми навчимося працювати з потоками в Node JS. Завдяки потокам ми можемо передавати дані поступово. Наприклад, ми можемо відображати лише шматочки даних, які вже отримали і в той же час отримувати іншу частину одного великого об'єкта.

Потоки в Node JS це відмінний засіб, яке дозволяє передавати величезні дані поступово, невеликими частинами. Уявіть собі величезну карту світу, яка завантажується не відразу, а лише шматочками. Таким чином користувач відразу зможе переглядати окрему її частину, поки інша буде ще довантажуватися.

Якщо ми маємо великий файл, то користувачеві не доведеться чекати поки він повністю завантажитися, так як окремі його частини будуть доступні практично відразу ж.

Для роботи з потоками необхідно використовувати модуль *fs*, з яким ми вже не раз працювали. У ньому існують методи createReadStream і createWriteStream, що дозволяють читати і записувати дані в файли.

Для виведення даних можна використовувати подію під назвою «*data*». У події можна виводити окремі шматочки даних як тільки вони будуть передані і готові до перегляду. Розглянемо приклад практичного застосування:

```
var fs = require('fs');

var myReadShort = fs.createReadStream(__dirname + '/article.txt', 'utf8');
var myWriteShort = fs.createWriteStream(__dirname + '/news.txt');

myReadShort.on('data', function(chunk) {
  console.log("Нові дані отримані:");
  myWriteShort.write(chunk);
});
```

За допомогою відповідних методів ми потоками зчитуємо дані з файлу article.txt і записуємо їх в news.txt. Щоб побачити як саме текст розділяється на шматки ми виводимо його на екран, розділяючи словами «Нові дані отримані:».

Маршрутизація Node JS

В даній темі ми навчимося обробляти посилання і виводити різні HTML-сторінки. Сторінки будуть виводитися в залежності від певної URL-адреси, що була введена в рядок браузера.

У минулих уроках ми вже створювали локальний сервер на основі модуля http. Також ми вже вміємо виводити дані безпосередньо на сайт. Залишилося лише відстежувати URL-адресу та в залежності від неї виводити різну інформацію на сторінку.

Для цього існує параметр request (запит), який містить значення url (адресу сторінки на даний момент). Все що необхідно робити, так це перевіряти дане значення і в залежності від нього видавати інформацію користувачеві.

Приклад відслідковування URL:

```
var server = http.createServer(function(req, res) {
    // В req.url знаходиться поточна адреса сторінки
    if (req.url === '/') {
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        fs.createReadStream(__dirname + '/index.html').pipe(res);
    } else if (req.url === '/news/best/articles') {
        res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        fs.createReadStream(__dirname + '/news-best.html').pipe(res);
    }
});
```

Ми перевіряємо «req.url» і при цьому використовуємо три знака рівності щоб перевірити не тільки на однакові значення, але і на однаковий

тип даних. Таким чином ми можемо перевіряти будь-яку URL-адресу та видавати ту сторінку, що буде підходити під потрібну адресу.

Подібна реалізація відстеження URL може здатися складною і зовсім нечитабельною. Це дійсно так, тому в курсі ми додатково вивчатимемо бібліотеку Express, яка вирішує багато незрозумілих моментів в Node. При використанні Express ви помітите, що відстежувати URL буде набагато простіше і логічніше.

В прикладі нижче ми підключаємо декілька створених HTML-сторінок до локального серверу і вводимо перевірку URL-адрес для них. Для випадку, якщо сторінка недоступна, виконується умова речення else, тобто сторінка з назвою 404. Щоб перейти на необхідну сторінку ми вказуємо її назву після адреси серверу у вікні браузера.

```
var fs = require('fs');

// Підключення до локального серверу

var http = require('http');

var server = http.createServer(function(req, res)
 {
 console.log("URL сторінки: " + req.url);
 if (req.url === '/index' || req.url === '/')
  {
  res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
  fs.createReadStream(__dirname + '/index.html').pipe(res);
  } else if (req.url === '/about')
  {
  res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
  fs.createReadStream(__dirname + '/about.html').pipe(res);
  } else {
// Сторінка для виключень
  res.writeHead(404, {'Content-Type': 'text/html; charset=utf-8'});
  fs.createReadStream(__dirname + '/404.html').pipe(res);
  }
 }
```

```
);

server.listen(3000, '127.0.0.1');
console.log("Ми відслідковуємо порт 3000");
```

Даний код доступний для копіювання, а нижче представлено код HTML-сторінок які також необхідно для цього створити.

Сторінка about.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Сторінки з інформацією</title>
 <style>
  body {background-color: #caddda;}
  .block {text-align: center; width: 70%; margin-left: 15%}
  h1 {color: #494949; font-size: 3.5em}
  p {color: #494949; font-size: 1.2em}
 </style>
</head>
<body>
 <div class="block">
  <h1> Сторінка з інформацією</h1>
  <p>Тут ви можете прочитати трохи інформації,
   а також цей текст, що повністю не має сенсу,
   але зате красиво написаний!</p>
  <p>КНУБА 2020</p>
 </div>
</body>
</html>
```

Сторінка 404.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Помилка 404</title>
 <style>
  body {background-color: #8c9aa1;}
  .block {text-align: center; width: 70%; margin-left: 15%}
  h1 {color: white; font-size: 3.5em}
  p {color: #fafafa; font-size: 1.2em}
 </style>
</head>
<body>
 <div class="block">
  <h1>Помилка 404!</h1>
  <p>Сторінку не знайдено!</p>
 </div>
</body>
</html>
```

Контрольні запитання

1. Що собою являє директива require()?
2. Як відбувається відстеження подій у Node JS?
3. Як считувати файли та працювати з директоріями у Node JS?
4. Як відбувається створення сервера на Node JS?
5. Як відбувається робота з потоками та маршрутизація Node JS?

Лекція 3. ВИКОРИСТАННЯ ПАКЕТНОГО МЕНЕДЖЕРА NPM

Node JS має потужний пакетний менеджер, який дозволяє встановлювати різні доповнення та вивантажувати власний код. Коли ви встановлюєте Node JS на комп'ютер то також автоматично встановлюєте пакетний менеджер npm (node package manager).

Менеджер дозволяє встановлювати бібліотеки в проект, створювати зв'язки між фреймворками, а також вивантажувати свій власний код в загальний доступ. Для роботи необхідно зайти в командний рядок (термінал) і прописати команди для установки пакетів через npm. Для установки всіх пакетів пропишіть команду: *npm install.* Для видалення пакетів використовуйте команду *npm uninstall* і після цього вказується назва пакету. Наприклад: npm uninstall express.

При відправці будь-якого файлу з кодом, для його коректної роботи можуть знадобитись певні бібліотеки. Файл package.json зберігає в собі саме ті бібліотеки, що зв’язані з проектом. Щоб його створити пропишіть команду *npm init* і впишіть всі необхідні дані через термінал. Після цього файл буде зберігати бібліотеки, для яких ви вкажете слово save.

Наприклад, щоб модуль express був зв’язаний з проектом і автоматично зберігся при його встановленні запишемо: npm install express **-save**.

Таким чином, отримавши ваші файли, інший користувач просто пише в командний рядок *npm install* і бібліотеки встановляться.

Вивчення фреймворка Express

В темі ми приступаємо до вивчення додаткової бібліотеки, яка називається Express. Встановимо бібліотеку, а також навчимося коректно обробляти різні URL з адресного рядка в браузері.

Бібліотека Express не є обов'язковою, проте, вона сильно спрощує роботу з сервером, відстеження посилань і додає можливість роботи з різними шаблонізаторами, про які ми поговоримо в наступних темах.

Щоб встановити бібліотеку введіть в терміналі команду *npm install express*. Далі бібліотека буде додана в ваш проект і ви зможете підключити її як звичайний модуль. Приклад підключення:

```
// Підключення бібліотеки
var express = require('express');
// Виклик основної функції, яка передається з модуля
var app = express();
```

Після підключення ми маємо доступ до всіх значень і змінним з модуля express. Для відстеження сервера і порту необхідно використовувати метод listen:

```
app.listen(3000); // відслідковування порта 3000
```

Щоб відстежувати посилання необхідно використовувати HTTP-запити. Їх існує декілька:

- get - отримання переходу на сторінку;
- post - отримання даних з форми;
- delete - отримання даних для видалення будь-якого об'єкта з БД;
- put - отримання даних для редагування будь-якого об'єкта з БД.

В ході уроку ми будемо працювати лише з get, так як нам необхідно лише відслідковувати переходи по конкретним посиланням. Наприклад, щоб відстежити посилання «/posts/best» можна скористатися наступним кодом:

```
app.get('/posts/best', function(req, res) {
    res.send('Тут відображаються кращі пости');
});
```

Для виведення інформації використовується метод send. Тобто коли ви відкриєте дану сторінку, на ній буде відображено текст, записаний в даний метод.

Використання шаблонізатора

Шаблонізатори дозволяють отримувати дані з мов програмування і виводити їх разом з розміткою. В цій темі ми навчимося працювати з шаблонізатором і виведемо дані через нього.

Шаблонізатори корисні і існують у багатьох мовах програмування. Їх основне завдання - передача даних з мови програмування в HTML-шаблон.

Отримавши дані ми можемо оперувати з ними як зі звичайними змінними: порівнювати, виводити на екран, додавати до них значення і так далі. Для Node JS існує шаблонізатор *EJS*, який виконує завдання на відмінно.

Для роботи з ним виконайте його підключення і встановіть пакет в проект. Для цього використовуйте команду *npm install ejs*. В головному файлі вкажіть шаблонізатор що ви використовуєте:

```
var express = require('express');
var app = express();
// Вказуємо шаблонізатор
app.set('view engine', 'ejs');
```

Всі файли-шаблони повинні зберігатися в папці views. Для відображення шаблонів на сторінці використовуйте метод *render* ().

```
app.get('/', function(req, res) {
    // Вивід HTML-шаблона "index"
    res.render('index', {someInfo: "Дані для передачі"});
});
```

Для створення шаблону необхідний файл з розширенням .ejs. У нього помістіть звичайний HTML-код і в певних місцях використовуйте розмітку шаблонізатора. Для використання спеціальної розмітки розміщуйте код в блоках: <%= %>. Більш детально про різноманітне виведення даних ви можете прочитати на офіційному сайті EJS - https://ejs.co/.

Розглянемо шаблонізатор на прикладі: Спочатку створимо файл так само, як ми це вже робили раніше, але з підключенням сторінки яка має змінні параметри і вказуємо які значення вони приймають. Використовуючи метод *render* ми передаємо шаблону ці параметри та відображаємо його в браузері при переході по вказаній адресі.

```
var express = require('express');

var app = express();

app.set('view engine', 'ejs'); //Вказуємо шаблонізатор

app.get('/news/:id', function(req, res) {
```

```
        var obj = {title: "Новина", id: 4, paragraphs: ['Параграф', 'Просто текст', 'Числа: 2, 4, 6', 99]};
        res.render('news', {newsId: req.params.id, newParam: 234, obj: obj});
    });

    app.listen(3000);
```

Тепер переходимо до іншого файлу - news.ejs, який зберігає в собі звичайний HTML-код нашої сторінки новин, але з деяким доповненням у вигляді вставок <% %>, що передає певну властивість у текстовий рядок. Таким чином ми передаємо newsId, title, масив paragraphs з об'єкту obj та ін. Його код:

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Новости</title>
      <style>
        body {background-color: #98bfd5;}
        .block {text-align: center; width: 70%; margin-left: 15%}
        h1 {color: #494949; font-size: 3.5em}
        p {color: #494949; font-size: 1.2em}
      </style>
    </head>
    <body>
      <div class="block">
        <h1>Сторінка новин з ID - <%= newsId %></h1>
        <p><b>Title:</b> <%= obj.title %></p>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.</p>
```

```
    <ul>
      <% obj.paragraphs.forEach(function(item) { %>
        <li><%= item %></li>
      <% }); %>
    </ul>
  </div>
</body>
</html>
```

Статичні файли та проміжне ПЗ

Щоб підключити статичні файли необхідно використовувати проміжне ПЗ. В темі ми познайомимося з підключенням статичних файлів, а також навчимося вбудовувати одні файли всередину інших.

Повторювані блоки сайту зручно виносити в окремі файли і викликати їх на всіх сторінках. Таким чином ми можемо змінити лише один файл і зміни будуть застосовані на всіх сторінках пов'язаних з блоком. Для створення блоків створіть файл і використовуйте конструкцію *include* для підключення файлу в HTML-шаблони:

```
<% include путь_к_файлу %>
```

Для роботи зі статичними файлами необхідно прописати метод, в якому визначається url-адреса і папка зі статичними файлами:

```
app.use('/public', express.static('public'));
```

Таким чином записуючи посилання, яке починається з *public* ми звертаємося до статичної папки, з ідентичною назвою в проекті і беремо звідти всі необхідні дані. Для наочного прикладу додаємо до коду з минулої теми рядки:

```
app.use('/public', express.static('public'));

app.get('/', function(req, res)
```

```
{
  res.render('index');
}
);
```

Тепер створимо файл index.ejs для цієї сторінки, код якого представлений нижче. Аналогічно прикладу з минулої теми підключаємо модулі (блоки), які мають відображатись на декількох сторінках, наприклад блок навігації, або стиль сторінок. Модулі створюються завчасно в окремому файлі (в папці public).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Заголовок сторінки</title>
  <link rel="stylesheet" href="/public/css/style.css">
</head>
<body>
  <% include blocks/header.ejs %>
  <div class="block">
    <h1>Чудовий сайт на Node JS</h1>
    <p> Node або Node.js - програмна платформа, заснована на
движку V8 (здійснює трансляцію JavaScript в машинний код), що
перетворює JavaScript з вузькоспеціалізованої мови в мову загального
призначення. Node.js додає можливість JavaScript взаємодіяти з
пристроями введення-виведення через свій API (написаний на C++),
підключати інші зовнішні бібліотеки, написані на різних мовах,
забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується
переважно на сервері, виконуючи роль веб-сервера, але є можливість
розробляти на Node.js і десктопні віконні додатки (за допомогою
NW.js, AppJS або Electron для Linux, Windows і Mac OS) і навіть
програмувати мікроконтролери (наприклад , tessel і espruino). В основі
```

```
Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням/висновком.</p>
        </div>
    </body>
    </html>
```

Створення HTML-форми та отримання даних

Node JS дозволяє коректно обробляти різні форми, перевіряти дані і взаємодіяти з ними. Для обробки існує спеціальний пакет body-parser. Після установки через пакетний менеджер і підключення до проекту як окремий модуль, ми можемо почати приймати дані з різних форм.

Спершу необхідно створити форму, при цьому додати до кожного поля атрибут name, так як без нього дані не будуть отримані з полів форми. У тезі <form> вкажіть метод відправки даних і url-адресу сторінки для обробки запиту.

У головному JS-файлі після підключення модуля body-parser використовуйте команду для створення змінної, яка буде отримувати усі дані з post-запиту:

```
var urlencodedParser = bodyParser.urlencoded({ extended: false });
```

Далі необхідно створити обробник post-запиту:

```
// Перевіряємо post-запит, замість звичного get
app.post('/contact', urlencodedParser, function(req, res) {
    // Якщо дані не передані, то повертаємо помилку
    if (!req.body) return res.sendStatus(400);
    // Всі дані з форми зберігаються в req.body
    console.log(req.body);
    // Можемо вивести іншу сторінку і передати в неї всі дані
    res.render('success', {data: req.body});
});
```

Створення сторінки точно таке ж, як і раніше. Ви можете передати на таку сторінку всі дані і взаємодіяти з ними як завгодно.

Модифікуємо наш приклад з минулих тем, щоб додати нову сторінку з post-запитом:

```
    var express = require('express');
```

```
var bodyParser = require('body-parser');

var app = express();

var urlencodedParser = bodyParser.urlencoded({ extended: false });

app.set('view engine', 'ejs');
app.use('/public', express.static('public'));

app.get('/', function(req, res) {
  res.render('index');
});

app.get('/about', function(req, res) {
  res.render('about');
});

app.post('/about', urlencodedParser, function(req, res) {
  if (!req.body) return res.sendStatus(400);
  console.log(req.body);
  res.render('about-success', {data: req.body});
});

app.get('/news/:id', function(req, res) {
  var obj = {title: "Новость", id: 4, paragraphs: ['Параграф',
'Обычный текст', 'Числа: 2, 4, 6', 99]};
  res.render('news', {newsId: req.params.id, newParam: 234, obj:
obj});
});

app.listen(3000);
```

Для коректної роботи всього проекту тепер нам не вистачає тільки двох файлів .ejs, тож створимо їх за прикладом нижче.

Файл about.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Сторінка з інформацією</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta.2/css/bootstrap.min.css"
 integrity="sha384-
PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povH
YgTpBfshb" crossorigin="anonymous">
  <style>
   body {background-color: #caddda;}
   .block {text-align: center; width: 40%; margin-left: 30%}
   h1 {color: #494949; font-size: 3.5em}
   p {color: #494949; font-size: 1.2em}
  </style>

</head>

<body>
  <% include blocks/header.ejs %>
  <div class="block">
   <h1>Сторінка з інформацією</h1>
   <p> Тут ви можете прочитати трохи інформації про нас, а
також прочитати цей
      текст, без сенсу, зате красиво написаний!</p>
   <form method="post" action="/about">
    <div class="form-group">
     <label for="exampleInputEmail1">Email address</label>
     <input type="email" name="email" class="form-control"
      id="exampleInputEmail1" aria-describedby="emailHelp"
```

```
                placeholder="Enter email">
                <small id="emailHelp" class="form-text text-muted">We'll
never share your
                email with anyone else.</small>
            </div>
            <div class="form-group">
              <label for="exampleInputPassword1">Password</label>
              <input type="password" name="pass" class="form-control"
                id="exampleInputPassword1" placeholder="Password">
            </div>
            <div class="form-check">
              <label class="form-check-label">
                <input type="checkbox" name="check" class="form-check-
input">
                Check me out
              </label>
            </div>
            <button type="submit" class="btn btn-
primary">Надіслати</button>
          </form>
        </div>
      </body>
      </html>
```

Файл about-success.ejs

```
      <!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Сторінки з інформацією</title>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta.2/css/bootstrap.min.css" integrity="sha384-
```

```
PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povH
YgTpBfshb" crossorigin="anonymous">
        <style>
         body {background-color: #caddda;}
         .block {text-align: center; width: 40%; margin-left: 30%}
         h1 {color: #494949; font-size: 3.5em}
         p {color: #494949; font-size: 1.2em}
        </style>
       </head>
       <body>
        <% include blocks/header.ejs %>
        <div class="block">
         <h1> Сторінка з інформацією </h1>
         <p> Тут ви можете прочитати трохи інформації про нас, а
також прочитати цей
              текст, без сенсу, зате красиво написаний!</p>
         <p>Дякуємо!</p>
         <p>
          <b>Email:</b> <%= data.email %><br>
          <b>Pass:</b> <%= data.pass %><br>
          <b>isCheck:</b> <%= data.check %>
         </p>
        </div>
       </body>
       </html>
```

Дані з URL

Ми підібралися до заключної теми по Node JS. Тут ми розглянемо як приймати дані з URL. Для цього необхідно використовувати параметр req і його значення query:

```
console.log(req.query);
```

Наприклад ми хочемо отримати дані з URL на сторінці news з нашого проекту. Це ті дані, що ми бачимо після назви сторінки (різні

фільтри, і т.д.). Виглядає це приблизно так: *.../news/some?filter=id&city=Kyiv*, і в консоль ми отримаємо дані про те, що filter = id, a city = Kyiv. Записується цей параметр наступним чином:

```
        app.get('/news/:id', function(req, res) {
         var obj = {title: "Новость", id: 4, paragraphs: ['Параграф',
'Обычный текст', 'Числа: 2, 4, 6', 99]};
         console.log(req.query);
         res.render('news', {newsId: req.params.id, newParam: 234, obj:
obj});
        });
```

На цьому ми завершуємо ознайомлення з Node JS. Ми розглянули всі необхідні моменти для роботи з даною платформою, тож тепер можна рухатись далі і вивчати тему більш детально самостійно.

Контрольні запитання

1. Як використовувати шаблонізатор у Node JS?
2. Що таке статичні файли та проміжне програмне забезпечення?
3. Як відбувається створення HTML-форми та отримання даних?
4. Як приймати дані з URL?

Лекція 4. EXPRESS

Знайомство з Express

Сайт Express характеризує Express як «мінімалістичний і гнучкий фреймворк для Node.js-веб-додатків, що забезпечує набір можливостей для побудови одно- і багатосторінкових і гібридних веб-додатків».

Що ж це насправді означає? це опис на складові частини.

1. Мінімалістичний.

Один з найбільш привабливих аспектів Express. Безліч раз розробники фреймворків забували, що зазвичай краще менше, та краще. Філософія

Express полягає в забезпеченні мінімальної прошарку між вашими мізками і сервером.

Це не означає ненадійність або недостатня кількість корисних можливостей. Просто він в меншій мірі стає у вас на шляху, дозволяючи вам більш повно висловлювати свої ідеї і в той же час забезпечуючи щось для вас корисне.

2. Гнучкий.

Іншим ключовим аспектом філософії Express є розширюваність. Express надає вам надзвичайно мінімалістичний фреймворк, і ви можете додавати функціональність в різні частини Express в міру необхідності, замінюючи все, що вам не підходить. Стільки фреймворків дають вам відразу все, залишаючи вас з роздутим, малозрозумілим, складним проектом ще до написання вами першого рядка коду. Дуже часто першим завданням стає чистка проекту від непотрібної функціональності або заміна функціональності, що не відповідає вашим вимогам.

Express реалізує протилежний підхід, дозволяючи додавати те, що вам потрібно, в міру необхідності.

3. Фреймворк для веб-додатків.

Ось тут семантика починає ускладнюватися. Що таке веб-додаток? Чи означає це, що ви не можете створювати сайти або веб-сторінки за допомогою Express?

Ні, сайт - це веб-додаток і вебсторінка - це веб-додаток. Але веб-додаток може бути чимось більшим: воно може забезпечувати функціональність для інших веб-додатків (поряд з іншими можливостями).

Взагалі слово «додаток» використовується для позначення чогось, у чого є функціональність: це не просто статичний набір контенту (хоча і це теж дуже простий приклад веб-додатки).

Зараз ще існує відмінність між додатком (чим-небудь запускаються нативної на вашому пристрої) і веб-сторінкою (чим-небудь переданим на ваш пристрій по Мережі), але ця різниця поступово розмивається завдяки таким проектам, як PoneGap, а також тому, що Microsoft дозволила виконувати HTML5-додатки на настільних комп'ютерах, як якщо б вони були нативними додатками.

Легко уявити, що через кілька років відмінність між додатком і сайтом взагалі зітреться.

4. Односторінкові веб-додатки.

Односторінкові веб-додатки - відносно нова ідея. На відміну від сайтів, які потребують виконання мережевого запиту кожного разу, коли користувач переходить на іншу сторінку, односторінкове веб-додаток викачує весь сайт (або солідний його шматок) в браузер клієнта. Після цієї первинного завантаження навігація прискорюється, оскільки взаємодія з сервером практично відсутня.

Розробка односторінкових веб-додатків полегшується завдяки використанню популярних фреймворків, таких як Anuar або Ember, які Express, на щастя, підтримує.

5. Багатосторінкові і гібридні веб-додатки.

Багатосторінкові вебпріложеній - більш традиційний підхід до сайтів. Кожна сторінка сайту забезпечується окремим запитом до сервера. Те, що цей підхід більш традиційний, не означає, що у нього немає переваг або що односторінкові веб-додатки в чомусь краще. Просто зараз з'явилося більше можливостей і ви можете самі вирішити, які частини контенту краще надати у вигляді односторінкового додатки, а які - за допомогою індивідуальних запитів.

Термін «гібридні» описує сайти, які реалізують обидва ці підходи. Якщо ви до цих пір не цілком розумієте, що таке Express насправді, не хвилюйтеся: іноді легше просто почати що-небудь використовувати, щоб зрозуміти, що це, і ця книга якраз допоможе вам почати створювати веб-додатки з Express і Node.

Поки, ймовірно, Node не справив на вас такого вже сильного враження. Ми, в попередній лекції по суті, повторили те, що Apace або IIS роблять для вас автоматично, однак тепер ви розумієте, як Node працює і наскільки ви можете їм управляти.

Так що тепер, коли у нас за плечима вже є невеликий досвід роботи з Node, ми готові перейти до вивчення Express.

Економія часу за допомогою Express

Ви вже дізналися, як створити простий веб-сервер за допомогою одного тільки Node. У цьому розділі ми відтворимо цей же сервер за допомогою Express, що дасть відправний пункт і познайомить вас з основами Express.

Скаффолдинг

Скаффолдинг – ідея проста: більшості проектів потрібна певна кількість так званого шаблонного коду, а кому хочеться заново писати цей код при створенні кожного нового проекту?

Простий спосіб вирішення проблеми - створити чорновий каркас проекту і кожен раз, коли потрібно новий проект, просто копіювати цей каркас, інакше кажучи, шаблон.

Шаблонний код також може принести користь для власне HTML, що видається клієнту. Я рекомендую вам чудовий HTML5 Boierpate (https: // html5boilerplate.com/). Він генерує відмінну «чисту дошку» для сайту на HTML5. Нещодавно в HTML5 Boierpate була додана можливість генерації користувальницької збірки. Один з варіантів користувальницької збірки включає Twitter Bootstrap - фреймворк для створення клієнтської частини, який рекомендується.

Перші кроки

Почнемо зі створення нового каталогу для вашого проекту - це буде його кореневої каталог. У даній книзі всюди, де ми говоримо про каталог проекту, каталозі додатки або кореневому каталозі проекту, ми маємо на увазі цей каталог.

Можливо, вам захочеться зберігати файли вашого веб-додатки окремо від усіх інших файлів, зазвичай супутніх проекту, таких як замітки з нарад, документація і т. П. Тому раджу зробити кореневим каталогом проекту окремий підкаталог того каталогу, в якому ви будете зберігати всю відповідну до проекту інформацію.

Протягом усієї лекції буде використовуватися єдиний приклад - вигаданий сайт турфірми Meadowlark Travel.

Наприклад, для сайту Meadowlark Travel я можу тримати проект в ~ / projects / meadowlark, а кореневим каталогом буде ~ / projects / meadowlark / site. Npm зберігає опис залежностей проекту - як і що відносяться до проекту метадані - в файлі package.json.

Найпростіший спосіб створити цей файл - виконати команду npm init: вона задасть вам ряд питань і згенерує package.json для початку роботи (на питання щодо точки входу (entry point) введіть meadowlark.js або використовуйте назву свого проекту).

Першим кроком буде **установка Express**.

Виконайте наступну команду npm:

npm install --save express

Виконання npm install встановить вказаний (-i) пакет (-и) в каталог node_modules.

Якщо ви вкажете прапор --save, файл package.json буде оновлено.

Оскільки каталог node_modules в будь-який момент може бути відновлений за допомогою npm, ми не станемо зберігати його в нашому репозиторії. Щоб переконатися, що ми не додали його випадково в репозиторій, створимо файл з ім'ям .gitignore:

Ігнорувати встановлені npm пакети: node_modules

Помістіть сюди будь-які інші файли, які ви не хочете вносити, такі як: DS_Store (OSX), * .bak, etc.

Тепер створимо файл meadowlark.js.

Це буде точка входу нашого проекту. Протягом книги ми будемо посилатися на цей файл просто як на файл програми:

```
var express = require ( 'express');

var app = express ();

app.set ( 'port', process.env.PORT || 3000);

// призначена для користувача сторінка 404 app.use (function (req, res) {

      res.type ( 'text / plain');

      res.status (404);

       res.send ( '404 - не знайдено'); });

// призначена для користувача сторінка 500 app.use (function (err, req, res, next) {

    console.error (err.stack);

      res.type ( 'text / plain');

       res.status (500);

       res.send ( '500 - Помилка сервера'); });

app.listen (app.get ( 'port'), function () {

  console.log ( 'Express запущений на http: // localhost:' + app.get ( 'port') + '; натисніть Ctrl + C для завершення.'); });
```

Багато керівництва, так само як і генератор скаффолдинг Express, закликають вас називати ваш основний файл app.js (іноді index.js або server.js).

Якщо ви не користуєтеся послугою хостингу або системою розгортання, що вимагає певного імені головного файлу програми, краще називати основний файл по назві проекту.

Той, хто колись вдивлявся в купу закладок редактора, які все називалися index.html, відразу ж визнає мудрість такого рішення. npm init за замовчуванням дасть ім'я index.js; якщо вам потрібно інше ім'я для файлу додатки, не забудьте змінити властивість main у файлі package.json.

Тепер у вас **є мінімальний сервер Express**.

Можете запустити сервер (node meadowlark.js) і перейти на http: // localhost 3000.

Результат буде невтішним: ви не надали Express ніяких маршрутів, так що він просто видасть вам узагальнену сторінку 404, яка вказує, що запитаної сторінки не існує.

Зверніть увагу на те, що ми вказали порт, на якому хочемо, щоб було запущено наш додаток: app.set (port, process.env.PORT || 3000). Це дозволяє перевизначити порт шляхом установки змінної середовища перед запуском сервера. Якщо ваше програма не запускається на порте 3000 при запуску цього прикладу, перевірте, чи встановлена змінна середовища PORT.

Вкрай рекомендується встановити плагін до браузера, який показував би вам код стану HTTP-запиту, так само як і будь-які відбуваються перенаправлення.

Це спростить виявлення проблем перенаправлення в вашому коді або неправильних кодів стану, які часто залишаються непоміченими. Для браузера Chrome чудово працює Redirect Path компанії Ayima. У більшості браузерів ви можете побачити код стану в розділі Мережа інструментів розробника.

Додамо маршрути для домашньої сторінки та сторінки Про ....

Перед оброблювачем 404 додаємо два нових маршрути:

```
app.get ( '/', function (req, res) {
    res.type ( 'text / plain');
    res.send ( 'Meadowlark Travel'); });
```

```
 app.get ( '/ about', function (req, res) {
       res.type ( 'text / plain');
       res.send ( 'Про Meadowlark Travel'); });
// призначена для користувача сторінка 404 app.use (function (req, res, next) {
       res.type ( 'text / plain');
      res.status (404);
      res.send ( '404 - не знайдено); });
```

app.get - метод, за допомогою якого ми додаємо маршрути.

У документації Express ви побачите app.VERB. Це не означає, що існує буквально метод з назвою VERB, це просто заповнювач для ваших (набраних в нижньому регістрі) HTTP-дієслів (найбільш поширені - et і post).

Цей метод приймає два параметри: шлях і функцію. Шлях - то, що визначає маршрут. Зауважте, що app.VERB виконує за вас важку роботу, а саме: за замовчуванням він ігнорує регістр і косу риску в кінці рядка, а також не бере до уваги рядок запиту під час виконання порівняння.

Так що маршрут для сторінки Про ... буде працювати для / about, / About, / about /, / about? Foo = bar, / about /? Foo = bar і т. п.

Створена вами функція буде викликатися при збігу маршруту. Передані цієї функції параметри - об'єкти запиту та відповіді. А поки ми просто повертаємо звичайний текст з кодом стану 200 (в Express код стану за замовчуванням дорівнює 200 - необхідність вказувати його явно немає).

Замість використання низькорівневого методу Node res.end ми перейдемо на використання розширення від Express res.send. Ми також замінимо метод Node res.writeHead методами res.set і res.status.

Express також надає нам для зручності метод res.type, що встановлює заголовок Content-Type. Хоча можна, як і раніше, використовувати методи res.writeHead і res.end, робити це не потрібно і не рекомендується.

Зверніть увагу на те, що наші користувальницькі сторінки 404 і 500 повинні оброблятися трохи інакше. Замість використання app.get застосовується app.use.

App.use - метод, за допомогою якого Express додає проміжне ПО. Ми будемо розглядати проміжне ПО далі, а поки ви можете вважати його узагальненим обробником усього, що не збігається з маршрутом.

Це призводить нас до дуже важливого нюансу: в Express порядок додавання маршрутів та проміжного програмного забезпечення має значення.

Якщо ми вставимо обробник 404 перед маршрутами, домашня сторінка і сторінка Про ... перестануть функціонувати, натомість їх URL будуть приводити до сторінці 404.

Поки наші маршрути досить прості, але вони також підтримують метасимволу, що може викликати проблеми з визначенням порядку проходження.

Наприклад, якщо ми хочемо додати до сторінці Про ... підсторінки, такі як / about / contact і / about / directions, наступний код не буде працювати очікуваним чином: app.get ('/ about *', function (req, res) {/ / відправляємо контент ...}) app.get ('/ about / contact', function (req, res) {// відправляємо контент ...}) app.get ('/ about / directions', function (req, res) {// відправляємо контент ...})

У цьому прикладі обробники / about / contact і / about / directions ніколи не будуть досягнуті, оскільки перший обробник містить метасимвол в своєму шляху: / about *.

Express може розрізнити обробники 404 і 500 за кількістю аргументів, які приймаються їхніми функціями зворотного виклику. Помилкові маршрути будуть розглянуті докладніше далі.

А тепер ви можете знову запустити сервер і переконатися в працездатності домашньої сторінки та сторінки Про ....

До сих пір ми не робили нічого, що не могло б бути настільки ж легко виконано без Express, але Express вже надав нам деяку не зовсім тривіальну функціональність.

Згадайте, як в попередньому розділі доводилося приводити req.url до єдиного вигляду, щоб визначити, який ресурс був запитаний? Нам довелося вручну прибрати рядок запиту і косу риску в кінці рядка, а також перетворити до нижнього регістру. Маршрутизатор Express тепер обробляє ці нюанси автоматично. Хоч зараз це може здатися не такою вже важливою річчю, це тільки верхній шар того, на що здатний маршрутизатор Express.

Уявлення і макети

В рамках парадигми «модель - представлення - контролер».

По суті, уявлення - то, що видається користувачеві. У разі сайту це зазвичай означає HTML, хоча ви також можете видавати PNG або P, або що-небудь ще, що може бути визуализировано клієнтом.

Для наших цілей будемо вважати, що уявлення - це HTML. Відмінність подання від статичного ресурсу, такого як зображення або файл CSS, - в тому, що уявлення не повинно бути статичним: HTML може бути створений на льоту для формування персоналізованої сторінки для кожного запиту.

Express підтримує безліч різних механізмів уявлень, які забезпечують різні рівні абстракції.

Рекомендується використовувати інший, менш абстрактний фреймворк шаблонізації - Handebars.

Handebars, заснований на популярному незалежному від мови програмування мовою шаблонізації Mustace, не намагається абстрагувати HTML: ви пишете HTML за допомогою спеціальних тегів, що дозволяють Handebars впроваджувати контент.

Щоб забезпечити підтримку Handebars, ми будемо використовувати пакет expresshandlebars, створений Еріком Феррайоло.

Виконайте наступне у вашому каталозі проекту:

```
npm install --save express-handlebars
```

Потім після створення додатка додайте в meadowlark.js наступні рядки:

```
var app = express (); // Установка механізму подання handlebars var handlebars = require ( 'express-handlebars').

create ({defaultLayout: 'main'});

app.engine ( 'handlebars', handlebars.engine);

app.set ( 'view engine', 'handlebars');
```

Це створює механізм подання і налаштовує Express для його використання за замовчуванням.

Тепер створимо каталог views з підкаталогом layouts.

Якщо ви досвідчений веб-розробник, то, ймовірно, вже добре знайомі з поняттям макетів (іноді також званих шаблонами сторінок).

Коли ви створюєте сайт, певні фрагменти HTML одні і ті ж - або майже одні й ті ж - на кожній сторінці. Переписувати весь цей повторюваний код на кожній сторінці не тільки втомлює, але і може послужити причиною справжнього кошмару при супроводі: якщо ви захочете змінити щось на кожній сторінці, вам доведеться змінити всі файли.

Макети звільняють вас від цієї необхідності, забезпечуючи загальний фреймворк для всіх сторінок вашого сайту.

Отже, створимо шаблон для нашого сайту. Створіть файл views / layouts / ma in.handlebars: <! Doctype html>

<Html> <head>

<Title> Meadowlark Travel </ title>

</ Head>

<Body>

{{{Body}}}

</ Body>

</ Html>

Єдине, чого ви, ймовірно, до сих пір не бачили, - це {{{body}}}. Цей вислів буде заміщено HTML-кодом для кожного уявлення.

Зверніть увагу на те, що при створенні екземпляра Handebars ми вказуємо макет за замовчуванням (defaultLayout: 'main').

Це означає, що, якщо ви не вкажете інше, для будь-якого уявлення буде використовуватися цей макет.

Тепер створимо сторінки уявлення для нашої домашньої сторінки, views / home.handlebars:

<H1> Ласкаво просимо в Meadowlark Travel

</ H1>

Потім для сторінки Про ..., views / about.handlebars:

<H1> Про Meadowlark Travel

</ H1>

Потім для сторінки не знайдено, views / 404.handlebars:

<H1> 404 - не знайдено

</ H1>

І нарешті, для сторінки Помилка сервера, views / 500.handlebars:

<H1> 500 - Server Error

</ H1>

Ймовірно, ви захочете, щоб ваш редактор асоціював .handlebars і .hbs (інше поширене розширення для файлів Handlebars) з HTML, щоб мати можливість використовувати підсвічування синтаксису та інші можливості редактора.

Що стосується vim, можете додати рядок:

au BufNewFile, BufRead * .handlebars set file type = html

в ваш файл ~ / .vimrc.

У разі використання іншого редактора загляньте в його документацію.

Тепер, коли ми встановили уявлення, необхідно замінити старі маршрути новими, які використовують ці уявлення:

```
app.get ( '/', function (req, res) {
   res.render ( 'home'); });
 app.get ( '/ about', function (req, res) {
   res.render ( 'about'); });
// Узагальнений обробник 404 (проміжне ПО)
 app.use (function (req, res, next) {
   res.status (404);
   res.render ( '404'); });
// Оброблювач помилки 500 (проміжне ПО)
app.use (function (err, req, res, next) {
   console.error (err.stack);
  res.status (500);
  res.render ( '500'); });
```

Зауважимо, що нам більше не потрібно вказувати тип контенту або код стану: механізм подання буде повертати за замовчуванням тип контенту text / html і код стану 200.

В узагальненому обработчике, що забезпечує призначену для користувача сторінку 404, і обработчике помилки 500 нам доводиться встановлювати код стану явно.

Якщо ви запустите свій сервер і перевірите домашню сторінку або сторінку Про ..., то побачите, що уявлення були візуалізовані.

А якщо подивитеся на вихідний код, виявите, що там є шаблонний HTML з views / layouts / main.handlebars.

Статичні файли та подання

При обробці статичних файлів і уявлень Express покладається на проміжне ПО. Проміжне ПО - поняття, яке буде докладніше розглянуто далі.

Поки ж вам достатньо знати, що проміжне ПО забезпечує розбиття на модулі, спрощуючи обробку запитів.

Проміжне ПО static дозволяє вам оголошувати один з каталогів як містить статичні ресурси, які досить прості для того, щоб їх можна було видавати користувачу без будь-якої особливої обробки.

Саме сюди ви можете помістити картинки, файли CSS і клієнтські файли на JavaScript.

У своєму каталозі проекту створіть підкаталог з ім'ям public (ми назвали його так, оскільки всі в ньому буде видаватися без будь-яких додаткових питань).

Потім перед оголошенням маршрутів додайте проміжне ПО static: app.use (express.static (__ dirname + '/ public')).

Проміжне ПО static призводить до того ж результату, що і створення для кожного статичного файлу, який ви хочете видати, маршруту, що візуалізує файл і повертає його клієнту.

Так що створимо підкаталог img всередині каталогу public і помістимо туди наш файл logo.png.

Тепер ми просто можемо послатися на /img/logo.png (зверніть увагу: ми не вказуємо public - цей каталог невидимий для клієнта), і статичну проміжне ПО видасть цей файл, встановивши відповідний тип контенту.

Виправимо наш макет таким чином, щоб логотип з'являвся на кожній сторінці: <body>

<Header>

<Img src = "/ img / logo.png" alt = "Логотип Meadowlark Travel">

</ Header>

{{{Body}}}

</ Body>

Елемент <header з'явився в HTML5 для забезпечення додаткової семантичної інформації про контент, що з'являється у верхній частині сторінки, такому як логотип, текст заголовка або інформація про навігації.

Динамічний контент в уявленнях. Уявлення - не просто ускладнений спосіб видачі статичного HTML (хоча вони, звичайно, можуть робити і це). Справжня міць уявлень в тому, що вони можуть містити динамічну інформацію.

Припустимо, ми хочемо на сторінці Про ... видавати віртуальні печива-передбачення. Визначимо в файлі meadowlark.js масив печива-пророкувань:

var fortunes = ["Победи свої страхи, або вони переможуть тебе.", "Рікам потрібні витоки.", "Не бійся невідомого.", "Тебе чекає приємний сюрприз.", "Будь простіше скрізь, де тільки можна.",] ;

Змініть уявлення (/views/about.handlebars) для відображення прогнозів:

<H1>

Про Meadowlark Travel

</ H1>

<P> Твоє пророцтво на сьогодні:

</ P>

<Blockquote>

{{Fortune}}

</ Blockquote>

Тепер поміняємо маршрут / about для видачі випадкового печива-передбачення: app.get ('/ about', function (req, res) {

```
var randomFortune = fortunes [Math.floor (Math.random () * fortunes.length)];
res.render ( 'about', {
fortune: randomFortune});
});
```

Тепер якщо ви перезапустіть сервер і завантажте сторінку / about, то побачите випадкове пророцтво.

Шаблонізації надзвичайно корисна, і ми розглянемо її детально далі.

Резюме

Ми створили найпростіший сайт за допомогою Express.

Контрольні запитання

1. Як зекономити час за допомогою Express?
2. Що таке скаффолдинг?
3. Як відбувається подання статичних створення файлів?

Лекція 5. NODE: НОВИЙ РІЗНОВИД ВЕБ-СЕРВЕРА

Поєднання JavaScript, Node і Express - ідеальний вибір для команд веб-розробників, яким потрібен потужний, швидко розгортається стек технологій, в рівній мірі визнається як в співтоваристві розробників, так і у великих компаніях

JavaScript – найпопулярніший мову написання виконуваних на стороні клієнта сценаріїв. На відміну від Flash, він підтримується всіма основними браузерами. Це основна технологія створення безлічі зустрічаються вам в Інтернеті привабливих анімацій і переходів.

Практично неможливо не використовувати JavaScript, якщо потрібно домогтися сучасної функціональності на стороні клієнта. Єдина проблема з JavaScript - він завжди чутливий до незграбному програмування. Екосистема Node змінює це, надаючи фреймворки, бібліотеки і утиліти, що прискорюють розробку і заохочують написання хорошого коду.

У певному сенсі у Node багато спільного з іншими популярними веб-серверами, такими як розроблений Microsoft веб-сервер Internet Information Services (IIS) або Apache. Але цікавіше, в чому його відмінності, так що почнемо з цього. Аналогічно Express, підхід Node до веб-серверів надзвичайно мінімалістичний. На відміну від IIS або Apache, для освоєння яких можуть знадобитися багато років, Node дуже легкий в установці та

налаштування. Це не означає, що настройка серверів Node на максимальну продуктивність в реальних умовах експлуатації тривіальна, просто конфігураційні опції - простіше і ясніше.

Інша фундаментальна відмінність між Node і більш традиційними веб-серверами - однопоточні Node.

На перший погляд це може здатися кроком назад. Але, як виявляється, це геніальна ідея. Однопоточні надзвичайно спрощує задачу написання веб-додатків, а якщо вам потрібно продуктивність багатопотокового додатку, можете просто запустити більше примірників Node і, по суті, отримати переваги багатопоточності.

Якщо говорити про спосіб створення програмного забезпечення, то додатки Node більше схожі на додатки PHP або Ruby, ніж на додатки .NET або Java. Движок JavaScript, що використовується Node (V8, розроблений компанією Gooe), не тільки компілює JavaScript у внутрішній машинний код (подібно C або C ++), але і робить це прозорим образом1, так що з точки зору користувача код поводиться як чистий інтерпретується мова програмування. Відсутність окремого кроку компіляції зменшує складність обслуговування і розгортання: досить просто оновити файл JavaScript, і ваші зміни автоматично стануть доступні.

Інша захоплююче гідність додатків Node - воістину неймовірна незалежність Node від платформи. Це не перша і не єдина платформонезалежна серверна технологія, але в незалежності від платформи важливіше пропоноване різноманітність платформ, ніж сам факт її наявності.

Наприклад, ви можете запустити додаток .NET на сервері під управлінням Linux за допомогою Mono, але це дуже нелегке завдання.

Аналогічно можете виконувати PHP-додатки на сервері під керуванням Windows, але їх налаштування зазвичай не так проста, як на машині з Linux.

У той же час Node елементарно встановлюється у всіх основних операційних системах (Windows, OS X і Linux) і полегшує совместWindows, OS X і Linux) і полегшує совмест, OS X і Linux) і полегшує совместLinux) і полегшує совмест) і полегшує спільну роботу.

Для команд веб-розробників мішанина PC і комп'ютерів Macintos цілком звичайна. Певні платформи, такі як .NET, задають непрості завдання розробникам і дизайнерам клієнтської частини додатків, часто

використовують комп'ютери Macintos, що серйозно позначається на спільній роботі і продуктивності праці.

Сама ідея можливості підключення працює сервера на будь-якій операційній системі за лічені хвилини (або навіть секунди!) - мрія, що стала реальністю.

Екосистема Node

В серцевині стека, звичайно, знаходиться Node.

Це ПО, яке забезпечує виконання JavaScript на віддаленому від браузера сервері, що, в свою чергу, дозволяє використовувати фреймворки, написані на JavaScript, такі як Express.

Іншим важливим компонентом є база даних.

Всі веб-додатки, крім хіба що самих простих, зажадають бази даних, і існують бази даних, які краще, ніж інші, підходять екосистемі Node.

Нічого дивного в тому, що є інтерфейси для всіх провідних реляційних баз даних (MySQL, MariaB, PostreSQL, Orace, SQL Server): було б нерозумно нехтувати цими визнаними китами.

Однак наступ епохи розробки під Node дало поштовх новому підходу до баз даних - появи так званих NoSQL-баз даних. Ці NoSQL-бази даних коректніше було б називати документоорієнтованих базами даних або базами даних типу «ключ - значення». Вони реалізують більш простий з понятійної точки зору підхід до зберігання даних.

Таких баз даних безліч, але MonoB - одна з лідерів, і саме її ми будемо використовувати.

Оскільки створення працездатного сайту залежить відразу від декількох технологічних складових, наприклад, поєднання Mono, Express, Anuar і Node.

Обов'язковий компонент, звичайно, Node. Хоча є й інші серверні JavaScript контейнери, Node стає в даний час переважаючим. Express теж не єдиний доступний фреймворк веб-додатків, хоча він наближається до Node за поширеністю.

Два інших компоненти, зазвичай істотних для розробки веб-додатків, - сервер баз даних і шаблонизатор (шаблонизатор забезпечує те, що зазвичай забезпечують PHP, JSP або Razor, - здатність гладко з'єднувати висновок коду і розмітки).

Для двох останніх компонентів очевидних лідерів немає, так що тут, я вважаю, було б шкідливо накладати якісь обмеження.

Разом всі ці технології об'єднує JavaScript, тому доцільно називати це стеком JavaScript. У даній роботі це означає Node, Express і MonoB.

Отримання Node

Встановити Node в вашу операційну систему простіше простого. Команда розробників Node зробила чимало для гарантованих простоти і ясності процесу установки на всіх основних платформах. Інсталяція настільки проста, що, по суті, її можна представити у вигляді трьох елементарних кроків.

1. Зайти на домашню сторінку Node.
2. Натиснути велику зелену кнопку з написом Встановити.
3. Слідувати вказівкам. Для Windows і OS X буде завантажена програма установки, яка проведе вас по процедурі установки.

У Linux швидше буде, ймовірно, використовувати сістеLinux швидше буде, ймовірно, використовувати систему управління пакетами.

Якщо ви користувач Linux і хочете використовувати систему управління пакетами, переконайтеся, що прямуєте вказівок з вищезгаданої веб-сторінки. Багато дистрибутиви Linux устаLinux уста встановлять дуже давню версію Node, якщо ви не додасте відповідний репозиторій пакетів.

Можете також завантажити автономну програму установки, що може виявитися корисним, якщо ви поширюєте Node у своїй організації.

Використання терміналу

Всі приклади в цій книзі будуть припускати використання терміналу. У багатьох з утиліт, згаданих в цій книзі, є графічний інтерфейс, так що, якщо ви категорично проти використання терміналу, у вас є вибір, проте вам доведеться розбиратися самостійно.

Редактори

Вибір текстового редактора- це ваш основний робочий інструмент. У Windows є кілька непоганих безкоштовних варіантів. Як у TextPad, так і у Notepad ++ є свої прихильники.

Обидва вони - багатофункціональні редактори з неперевершеною (оскільки вона дорівнює нулю) ціною. Якщо ви користувач Windows, не ігноруйте Visua Studio як редактор JavaScript: вона виключно функціональна і володіє одним з кращих двигунів автодоповнення серед

усіх редакторів. Ви можете безкоштовно скачати Visua Studio Express з сайту Microsoft.

Npm

Npm - повсюдно поширена система управління пакетами для пакетів Node (саме таким чином ми отримаємо і встановимо Express). На відміну від PHP, GNU, WINE скоріше, це рекурсивна абревіатура для «npm - НЕ акронім».

Взагалі кажучи, двома основними завданнями системи управління пакетами є установка пакетів і управління залежностями.

Npm – швидка і ефективна система управління пакетами, якої, екосистема Node багато в чому зобов'язана своїм швидким ростом і різноманітністю.

Npm встановлюється при інсталяції Node, так що, якщо ви слідували пере встановлюється при інсталяції Node, так що, якщо ви слідували перерахованим раніше кроків, вона у вас вже є. Так приступимо ж до роботи!

Основна команда, яку ви будете використовувати з npm (що не дивно), - install. Наприклад, щоб встановити Grunt (популярний виконавець завдань для JavaScript), можна виконати наступну команду: npm install -g grunt-cli.

Прапор -g повідомляє npm про необхідність глобальної установки пакета, що означає його доступність по всій системі. Ця різниця буде зрозуміліше, коли ми розглянемо файли package.json. А поки приймемо за емпіричне правило, що утиліти JavaScript, такі як Grunt, зазвичай будуть встановлюватися глобально, а специфічні для вашого веб-додатки пакети - немає.

Простий веб-сервер за допомогою Node

Node пропонує парадигму, відмінну від парадигми звичайного веб-сервера: створюване вами додаток і є веб-сервером.

Node просто забезпечує вас фреймворком для створення веб-сервера.

Node робить написання цього веб-сервера найпростішим справою (іноді всього кілька рядків), і контроль, який ви натомість отримуєте над вашим додатком, більш ніж варто того. Зробимо це.

Hello World

У своєму улюбленому редакторі створіть файл під назвою helloWorld.js:

```
 var http = require ( 'http');
http.createServer (function (req, res) {
   res.writeHead (200, { 'Content-Type': 'text / plain'});
   res.end ( 'Hello world!'); }). Listen (3000);
console.log ( 'Сервер запущений на localhost: 3000;
натисніть Ctrl-C для завершення .... ');
```

переконайтеся, що ви перебуваєте в тій же директорії, що і helloWorld.js,

і наберіть node helloWorld.js.

Потім відкрийте браузер, перейдіть на http: // localhost: 3000, і - вуаля! - ваш перший веб-сервер.

Саме цей веб-сервер не видає HTML, скоріше, він просто передає повідомлення «Нео word!» У вигляді неформатированного тексту вашому браузеру.

Якщо хочете, можете поекспериментувати з відправкою натомість HTML: просто поміняйте text / plain на text / html і змініть 'Hello world!' на рядок, що містить коректний HTML.

Подієво-кероване програмування

Базовим принципом Node є подієво-кероване програмування. Для вас як програміста це означає необхідність розуміти, які події вам доступні і як реагувати на них.

Багато людей знайомляться з собитійноуправляемим програмуванням при реалізації призначеного для користувача інтерфейсу: користувач натискає на що-небудь, і ви обробляєте подія натискання.

Це хороша метафора, оскільки ясно, що програміст не керує тим, коли користувач натискає на щось і натискає взагалі, так що подієво-кероване програмування дійсно є цілком наочним.

Може виявитися трохи більш складним зробити уявний перехід до реагування на події на сервері, але принцип залишається тим же самим.

У попередньому прикладі коду подія є неявним: обробляється подія - HTTP-запит.

Метод http.createServer приймає функцію як аргумент, ця функція буде викликатися кожен раз при виконанні HTTP-запиту.

Наша проста програма просто встановлює в якості типу вмісту звичайний текст і відправляє рядок «Hello word!».

Маршрутизація

Маршрутизація відноситься до механізму видачі клієнту контенту, який він запитував.

Для клієнт-серверних веб-додатків клієнт визначає бажаний контент в URL, а саме шлях і рядок запиту.

Розширимо наш приклад з «Нео word!» так, щоб він робив щось цікавіше. Для цього створимо мінімальний сайт, що складається з домашньої сторінки, сторінки Про ... і сторінки не знайдено.

Поки будемо дотримуватися попереднього прикладу і просто станемо видавати звичайний текст замість HTML:

```
var http = require ( 'http');

http.createServer (function (req, res) {

  // Наводимо URL до єдиного вигляду шляхом видалення // рядка запиту, необов'язковою косою риси // в кінці рядка і приведення до нижнього регістру

 var path = req.url.replace (/\/?(?:\?.*)?$/, '') .toLowerCase ();

  switch (path) {

    case '':

        res.writeHead (200, { 'Content-Type': 'text / plain'});

        res.end ( 'Homepage');

        break;

    case '/ about':

        res.writeHead (200, { 'Content-Type': 'text / plain'});

        res.end (O ');

        break;

    default:

        res.writeHead (404, { 'Content-Type': 'text / plain'});

        res.end ( 'не знайдено');

        break;
```

}}). Listen (3000);

console.log ('Сервер запущений на localhost: 3000; натисніть Ctrl + C для завершення ....');

Якщо ви запустите це, то виявите, що тепер можете переходити на домашню сторінку (http: // localhost: 3000) і сторінку Про ... (http: // localhost: 3000 / about).

Будь-які рядки запиту будуть проігноровані, так що http: // localhost: 3000 /? foo = bar поверне домашню сторінку, а будь-який інший URL поверне сторінку не знайдено.

Видача статичних ресурсів

Тепер, коли у нас запрацювала найпростіша маршрутизація, видамо який-небудь справжній HTML і картинку логотипу.

Вони носять назву статичних ресурсів, оскільки не змінюються (на відміну від, наприклад, тікера: кожен раз, коли ви перевантажуєте сторінку, біржові котирування міняються).

Якщо ви працювали з Apache або IIS, то, ймовірно, звикли просто створювати HTML-файл і переходити до нього з автоматичною передачею його браузеру. Node працює по-іншому: нам доведеться виконати роботу по відкриттю файлу, його читання і потім відправку його вмісту браузеру.

Так що створимо в нашому проекті каталог public (в наступному розділі стане зрозуміло, чому ми не називаємо його static).

У цьому каталозі створимо файли home.html, about.html, 404.html, підкаталог з назвою img і зображення з ім'ям img / logo.jpg.

У ваших HTML-файлах посилайтеся на логотип наступним чином: <img src = "/ img / logo.jpg" alt = "logo">.

Тепер внесемо поправки в helloWorld.js:

```
var http = require ( 'http'), fs = require ( 'fs');

function serveStaticFile (res, path, contentType, responseCode) {

  if (! responseCode) responseCode = 200;

  fs.readFile (__ dirname + path, function (err, data) {

    if (err) {

      res.writeHead (500, { 'Content-Type': 'text / plain'});
```

```
            res.end ( '500 - Internal Error'); } Else {
            res.writeHead (responseCode, { 'Content-Type': contentType});
           res.end (data); }}); }
http.createServer (function (req, res) {
  // Наводимо URL до єдиного вигляду шляхом видалення
 // рядка запиту, необов'язковою косою риси
// в кінці рядка і приведення до нижнього регістру
    var path = req.url.replace (/\/?(?:\?.*)?$/, '') .toLowerCase ();
    switch (path) {
        case '':
              serveStaticFile (res, '/public/home.html', 'text / html');
              break;
       case '/ about':
               serveStaticFile (res, '/public/about.html', 'text / html');
               break;
        case '/img/logo.jpg':
              serveStaticFile (res, '/public/img/logo.jpg', 'image / jpeg');
              break;
       default:
              serveStaticFile (res, '/public/404.html', 'text / html', 404);
               break
   }}). Listen (3000);
console.log ( 'Сервер запущений на localhost: 3000; натисніть Ctrl + C для завершення ....');
```

У цьому прикладі ми показали не дуже багато винахідливості в питанні маршрутизації.

Якщо ви перейдете за адресою http: // localhost: 3000 / about, буде виданий файл public / about.html.

Можна змінити шлях і файл на будь-які, які вам тільки захочеться.

Наприклад, якщо у вас є окрема сторінка Про ... на кожен день тижня, у вас можуть бути файли public / about_mon.html, public / about_tue.html і т. Д., А логіка маршрутизації може бути зроблена такою, щоб видавати відповідну сторінку при переході користувача за адресою http: // localhost: 3000 / about.

Зверніть увагу на те, що тут ми створили допоміжну функцію,

serveStaticFile, що виконує масу роботи.

fs.readFile - асинхронний метод для читання файлів.

Існує синхронна версія цієї функції - fs.readFileSync, але чим швидше ви почнете мислити асинхронно, тим краще.

Ця функція проста: вона викликає fs.readFile для читання вмісту зазначеного файлу. Коли файл прочитаний, fs.readFile виконує функцію зворотного виклику; якщо файлу не існує або були проблеми з правами доступу при читанні файлу, встановлюється змінна err і функція повертає код стану HTTP 500, який вказує на помилку сервера.

Якщо файл був прочитаний успішно, він відправляється клієнту з заданим кодом відповіді і типом вмісту.

Контрольні запитання

1. Якою є екосистема Node?
2. Як встановити Node в вашу операційну систему?
3. Що таке Npm?
4. Як створити простий веб-сервер за допомогою Node?
5. Що таке подієво-кероване програмування?
6. Як відбувається маршрутизація та видача статичних ресурсів?

Список використаної літератури

1. Web Development with Node and Express, 2nd Edition by Ethan Brown

   Released November 2019. Publisher(s): O'Reilly Media, Inc. ISBN: 9781492053514

2. Документація Node.js [Електронний ресурс]. - Режим доступу: URL https://nodejs.org/uk/docs/

3. Документація Express.js [Електронний ресурс]. - Режим доступу: URL https://expressjs.com/ru/guide/routing.html

4. Електронний навчальний курс Гончаренко Т.А. Методичні вказівки до виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування» Київ: КНУБА, 2018 [Електронний документ]. Режим доступу http://do2.knuba.edu.ua/course/view.php?id=55

Навчально-методичне видання

КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Конспект лекцій

для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальностями
122 “Комп’ютерні науки”,
126 “Інформаційні системи і технології”

Укладачі: Гончаренко Тетяна Андріївна
Хроленко Володимир Миколайович
Голенков Володимир Геннадійович