

**МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота №2**

**з дисципліни “ Основи web-програмування ”**

**тема “ОСНОВИ JAVASCRIPT”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент II курсу**  **групи КП-83**  **Мортіков Владислав Євгенович** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

**Мета роботи**

Познайомитись з мовою програмування JavaScript та середовищем Node.js. Навчитись створювати модулі, синхронно працювати із файловими потоками та обробляти JSON-текст.

**Постановка завдання**

Реалізувати консольну програму для керування сутностями різних типів даних із файлового сховища у форматі JSON.

**Вказівки до виконання**

1. Підготувати Node.js проект
2. Встановити найновішу доступну LTS версію Node.js (node).
3. Встановити глобально пакет ESLint за допомогою команди: npm install -g eslint.
4. Створити у навчальному репозиторії директорію проекту: labs/lab2 і перейти до неї у редакторі (IDE).
5. Створити у корені проекту файл .eslintrc.json та app.js (вміст файлів взяти з Додатків даного завдання).
6. Додати у корінь проекту файл .gitignore (вміст файлу)
7. Відкрити термінал у кореневій директорії завдання і виконати команду: npm init --yes, яка автоматично створить файл package.json. Всі зміни закомітити у віддалений репозиторій.
8. Спеціальні директорії проекту: директорії models та data.

8.1. Сховище користувачів:

Створити модуль models/user.js (див. приклад) і підключити його у app.js:

// models/user.js

module.exports = {};

// app.js

const user = require('./models/user');

// …

Використати у модулі тип сутностей Користувач для створення модуля-сховища колекції об'єктів цього типу. Кожен користувач повинен містити такі поля:

id - унікальний числовий (або uuid) ідентифікатор

login - унікальний рядок символів

role - ціле число (0 - простий користувач, 1 - адміністратор).

fullname - рядок повного імені користувача

registeredAt - рядок із датою у форматі ISO 8601.

avaUrl - рядок з URL зображення.

isDisabled - відмітка чи користувача було деактивовано.

Модуль models/user.js повинен містити такі синхронні функції:

getAll() - отримати списком всі об'єкти зі сховища.

getById(id) - отримати елемент зі сховища за ідентифікатором. У випадку, якщо елемент не знайдено - повертати undefined.

Реалізувати модуль таким чином, щоби всі описані функції синхронно працювали із JSON файлом (data/users.json), що буде містити масив із об'єктами користувачів. Рекомендована структура файлу (див. Додатки).

Реалізувати у app.js командний інтерфейс користувача для виконання операцій з сутностями типу Користувач (код для отримання вводу з консолі у Додатках, формат команд можна змінювати):

На команду users завантажити і показати користувачу список (або таблицю) з короткою інформацією про всі сутності.

На команду users/get/{id} (замість {id} вводиться ідентифікатор сутності) вивести у консоль детальну інформацію про обрану сутність.

Сховище сутностей за варіантом:

Створити модуль models/{entity}.js (замість {entity} використати назву сутності англійською мовою) і підключити його у app.js. Використати у модулі тип сутностей із ЛР№1 для створення модуля-сховища колекції об'єктів цього типу.

Кожен об'єкт у сховищі повинен містити як мінімум:

унікальний ідентифікатор, що присвоюється йому при внесенні у сховище

2 рядкових значеня

2 числових значення

1 рядок із датою у форматі ISO 8601.

Модуль models/{entity}.js повинен містити такі синхронні функції:

insert(x) - додати у сховище новий елемент та повернути його новий ідентифікатор

getAll() - отримати списком всі об'єкти зі сховища

getById(id) - отримати елемент зі сховища за ідентифікатором

update(x) - оновити дані елемента у сховищі

deleteById(id) - видалити елемент зі сховища за ідентифікатором

Реалізувати модуль таким чином, щоби всі описані функції синхронно працювали із JSON файлом (data/{entities}.json, замість {entities} використати назву сутності англійською мовою у множині), що буде містити масив із об'єктами (структура аналогічна data/users.json).

Також розмістити у цьому файлі ідентифікатор останнього доданого об'єкту (nextId), і інкрементувати його при додаванні нових елементів у сховище для забезпечення унікальності ідентифікаторів колекції елементів.

Кожна зміна вмісту сховища повинна перезаписувати файл. Текст JSON зберігати із відступами (indented).

Реалізувати оброблення консольних команд, за допомогою яких користувач буде мати можливість використовувати всі функцій із модуля-сховища сутностей за варіантом, наприклад (формат команд можна змінювати):

{entities}/get/{id} - отримати детальну інформацію про обрану сутність

{entities}/delete/{id} - видалити зі сховища обрану сутність

{entities}/update/{id} - ввести нові дані для обраної сутності і оновити її у сховищі

{entities}/insert - розпочати заповнення даних нової сутності, яку додати у сховище

На початку роботи програми (головне меню) дати користувачу моживість переглянути всі сутності обраного типу за допомогою команд:

users - користувачі

{entities} - сутності за варіантом

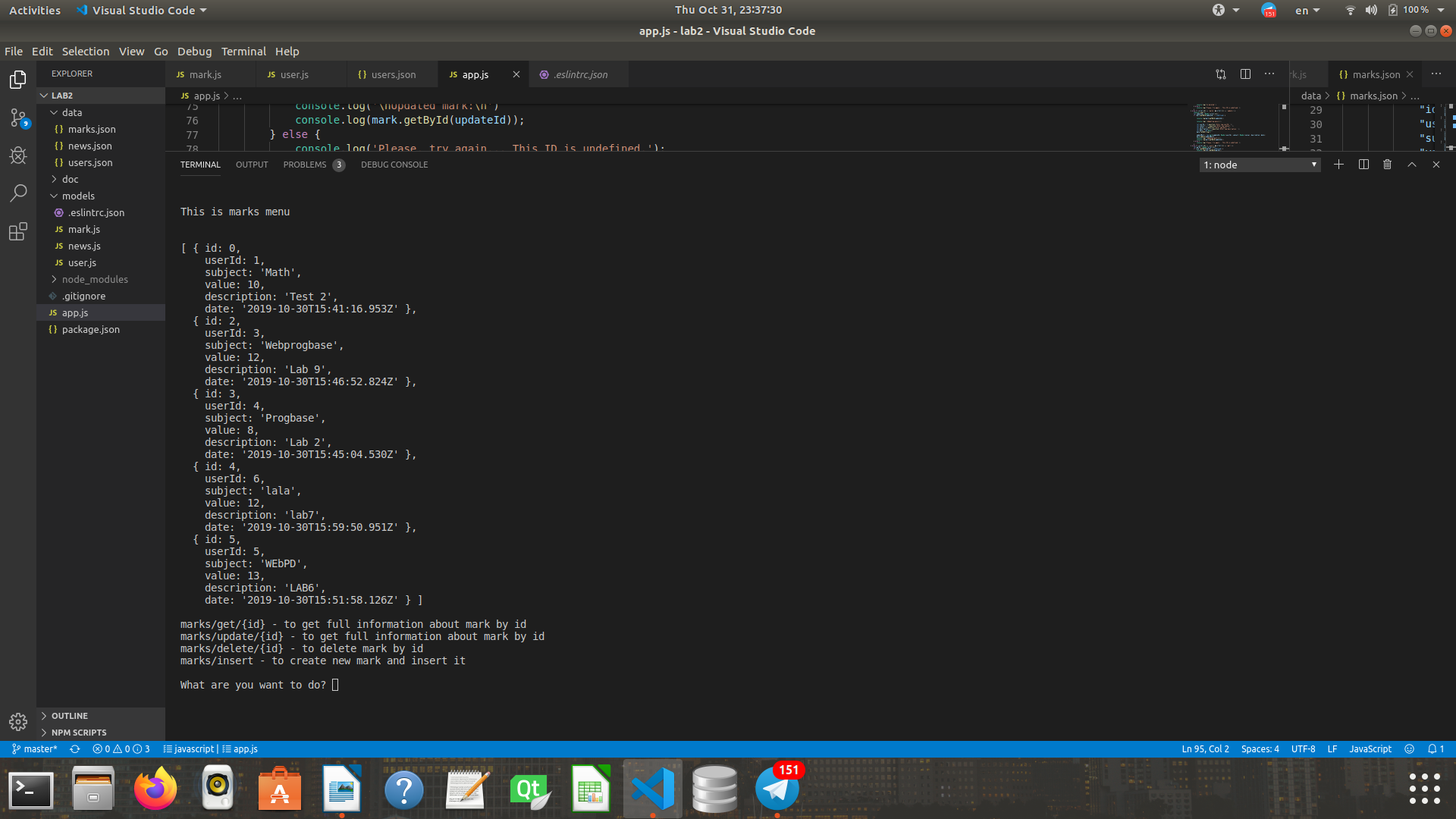
**Тексти коду програм**

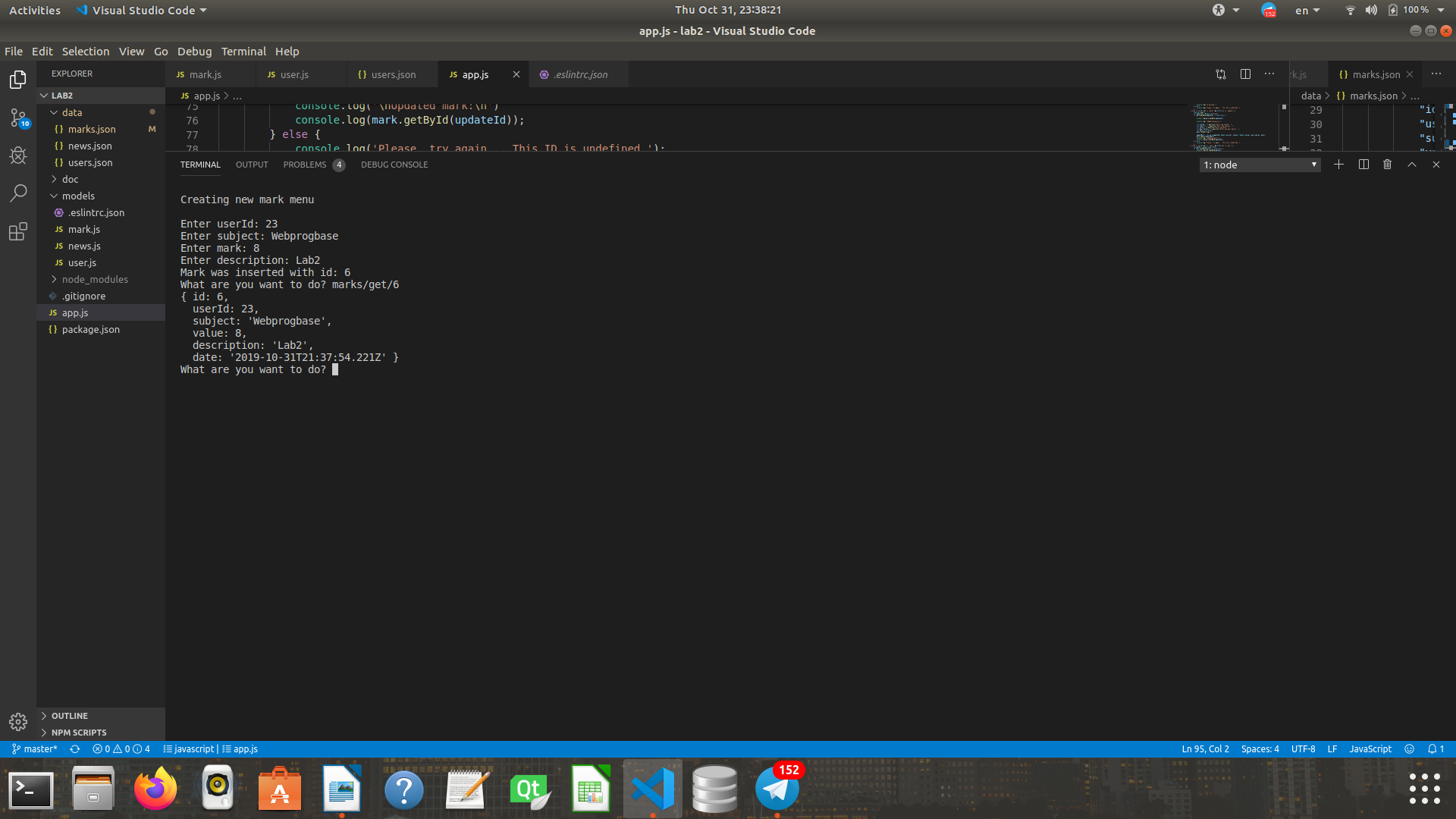
|  |
| --- |
| **user.js** |
| **const fileName = "./data/users.json";**  **const fs = require('fs');**  **class User {**  **constructor(id, login, role, fullName, registeredAt, avaUrl, isDisabled) {**  **this.id = id;**  **this.login = login;**  **this.role = role;**  **this.fullName = fullName;**  **this.registeredAt = registeredAt;**  **this.avaUrl = avaUrl;**  **this.isDisabled = isDisabled;**  **}**  **static getAll() {**  **let dataRead = fs.readFileSync(fileName, 'utf8');**  **return (JSON.parse(dataRead).items);**  **}**  **static getById(id) {**  **let elementById = read(fileName);**  **const user = elementById.items;**  **return user.find(x => x.id === id);**  **}**  **}**  **module.exports = User;** |

|  |
| --- |
| **mark.js** |
| **const marksFileName = "./data/marks.json";**  **const fs = require('fs');**  **class Mark {**  **constructor(id, userId, subject, value, description, date) {**  **this.id = id;**  **this.userId = userId;**  **this.subject = subject;**  **this.value = value;**  **this.description = description;**  **this.date = date;**  **}**  **static insert(mark) {**  **let dataRead = read(marksFileName);**  **mark.id = dataRead.nextId;**  **dataRead.items.push(mark);**  **dataRead.nextId++;**  **write(dataRead);**  **return mark.id;**  **}**  **static getAll() {**  **let data = fs.readFileSync(marksFileName, "utf-8");**  **return (JSON.parse(data).items);**  **}**  **static getById(id) {**  **let elementById = read(marksFileName);**  **const mark = elementById.items;**  **return mark.find(x => x.id === id);**  **}**  **static update(mark)**  **{**  **let element = read(marksFileName)**  **for (let i in element.items)**  **{**  **if (element.items[i].id == mark.id)**  **element.items[i] = mark;**  **}**  **write(element);**  **}**  **static deleteById(id) {**  **let data = read(marksFileName);**  **let deletedData;**  **for (let i = 0; i < data.items.length; i++) {**  **if (data.items[i].id === id) {**  **deletedData = data.items.splice(i, 1);**  **}**  **if (deletedData !== undefined) {**  **write(data);**  **}**  **}**  **return deletedData;**  **}**  **}**  **function read(path) {**  **let dataRead = fs.readFileSync(path, "utf-8");**  **return JSON.parse(dataRead);**  **}**  **function write(elements) {**  **let fileContent = JSON.stringify(elements, null, 4);**  **fs.writeFileSync(marksFileName, fileContent);**  **}**  **module.exports = Mark;** |

|  |
| --- |
| **app.js** |
| **const user = require('./models/user');**  **const news = require('./models/news');**  **const mark = require('./models/mark');**  **const rl = require('readline-sync');**  **while (1) {**  **let inputString = rl.question('What are you want to do? ')**  **let separator = '/';**  **let strArr = inputString.split(separator);**  **if (inputString == 'users') {**  **console.clear();**  **console.log('\n\nThis is users menu\n\n');**  **console.log(user.getAll());**  **} else if (inputString == 'marks') {**  **console.clear();**  **console.log('\n\nThis is marks menu\n\n');**  **console.log(mark.getAll());**  **console.log('\nmarks/get/{id} - to get full information about mark by id');**  **console.log('marks/update/{id} - to get full information about mark by id');**  **console.log('marks/delete/{id} - to delete mark by id');**  **console.log('marks/insert - to create new mark and insert it\n');**  **} else if (inputString == 'marks/insert') {**  **console.clear();**  **console.log('\nCreating new mark menu\n')**  **let userId = rl.question('Enter userId: ');**  **let subject = rl.question('Enter subject: ');**  **let value = rl.question('Enter mark: ');**  **let description = rl.question('Enter description: ');**  **let date = new Date();**  **date.toISOString();**  **insertMark = new mark(0, Number(userId), subject, Number(value), description, date);**  **let newId = mark.insert(insertMark);**  **console.log('Mark was inserted with id: ' + newId);**  **} else if (strArr[0] == 'marks' && strArr[1] == 'get') {**  **let getId = Number(strArr[2]);**  **if (mark.getById(getId) != undefined) {**  **console.log(mark.getById(getId));**  **} else {**  **console.log('Please, try again... This ID is undefined.');**  **}**  **} else if (strArr[0] == 'marks' && strArr[1] == 'delete') {**  **console.clear();**  **let deleteId = Number(strArr[2]);**  **if (mark.getById(deleteId) != undefined) {**  **mark.getById(deleteId);**  **mark.deleteById(deleteId);**  **console.log('Is deleted!')**  **} else {**  **console.log('Please, try again... This ID is undefined.');**  **}**  **} else if (strArr[0] == 'marks' && strArr[1] == 'update') {**  **console.clear();**  **let updateId = Number(strArr[2]);**  **if (mark.getById(updateId) != undefined) {**  **console.log(mark.getById(updateId));**  **console.log('\nUpdating menu\n');**  **let userId = rl.question('Enter new userId: ');**  **let subject = rl.question('Enter new subject: ');**  **let value = rl.question('Enter new mark: ');**  **let description = rl.question('Enter new description: ');**  **let date = new Date();**  **date.toISOString();**  **updateMark = new mark(updateId, Number(userId), subject, Number(value), description, date);**  **mark.update(updateMark);**  **console.log('\nUpdated mark:\n')**  **console.log(mark.getById(updateId));**  **} else {**  **console.log('Please, try again... This ID is undefined.');**  **};**  **} else if (strArr[0] == 'users' && strArr[1] == 'get') {**  **let getId = Number(strArr[2]);**  **if (user.getById(getId) != undefined) {**  **console.log(user.getById(getId));**  **} else {**  **console.log('Please, try again... This ID is undefined.');**  **}**  **} else if (inputString == 'exit') {**  **console.clear();**  **console.log('Bye!')**  **process.exit();**  **}**  **else {**  **console.log('Please, check your input and try again');**  **}**  **}** |

**Приклади результатів**





**Висновки**

Виконавши дану лабораторну роботу ми навчилися працювати з JavaScript, а саме познайомились з мовою програмування JavaScript та середовищем Node.js і навчились створювати модулі, синхронно працювати із файловими потоками та обробляти JSON-текст.

Компіляція всього коду відбувалася за допомогою node app.js .