Домашнє завдання №27_1

Застосовуючи елементи реактивної парадигми програмування(код має містити в явній формі Observable та Observer) за допомогою RxJS(реалізація ReactiveX для JavaScript) скласти програму (мовою JavaScript для платформи NodeJS), яка дозволяє виконувати валідацію ключа(25 шістнадцяткових цифр) ліцензії для деякого програмного продукту. Для зручності вводу програма має групувати шістнадцяткові цифри по п'ять цифр та відображає 'A', 'B', 'C', 'D', 'E' та 'F' завжди у верхньому регістрі. Окрім шістнадцяткових цифр та символів пробілу і табуляції(що трактуються як одна цифра 0), програма валідації не дозволяє вводити жодних інших символів, які не належать шістнадцятковій системі числення. При цьому надається не більше ATTEMPTS_COUNT спроб вводу ключа ліцензії. Між ітераціями сусідніх спроб введені значення мають зберігатися для редагування у наступній спробі.

Ключ ліцензії «11111 22222 33333 44444 55555» має міститися в коді програми валідації у неявній формі.

** акцентується увага на тому, що завдання має бути виконано за допомогою RxJS відповідно до парадигми реактивного програмування, а не подійно-орієнтованої парадигми, на основі якої працює рушій платформи NodeJS*

** **коментар:** це завдання буде аналогічне завданням №28_1, №28_2 та №28_3 і є повністю тотожне до завдань №27_2 та №27_3; таким чином можна порівняти різні засоби програмування; далі наводиться приклад повністю виконаного завдання; для компіляції і запуску можна використати https://repl.it/languages/nodejs , нижче також показаний спосіб виконання наведеного прикладу коду зо допомогою цього засобу .*

Вибір варіанту

(Nж + Nг + 1) % 10 + 1

де: Nж – порядковий номер студента в групі, а Nг – номер групи(1,2,3,4,5,6,7,8 або 9)

Варіанти завдань

Номер **варіанту** відповідає максимально допустимій кількості спроб(ATTEMPTS_COUNT) для введення ключа ліцензії.

Спосіб виконання наведеного прикладу коду за допомогою https://repl.it/languages/nodejs

На рисунках 1, 2, 3, 4 та 5 послідовно показаний спосіб виконання наведеного прикладу коду за допомогою https://repl.it/languages/nodejs . При додаванні пакету RxJS

потрібно дочекатися завершення процесу додавання. На рисунках 6 та 7 показані спроби введення ключа ліцензії.

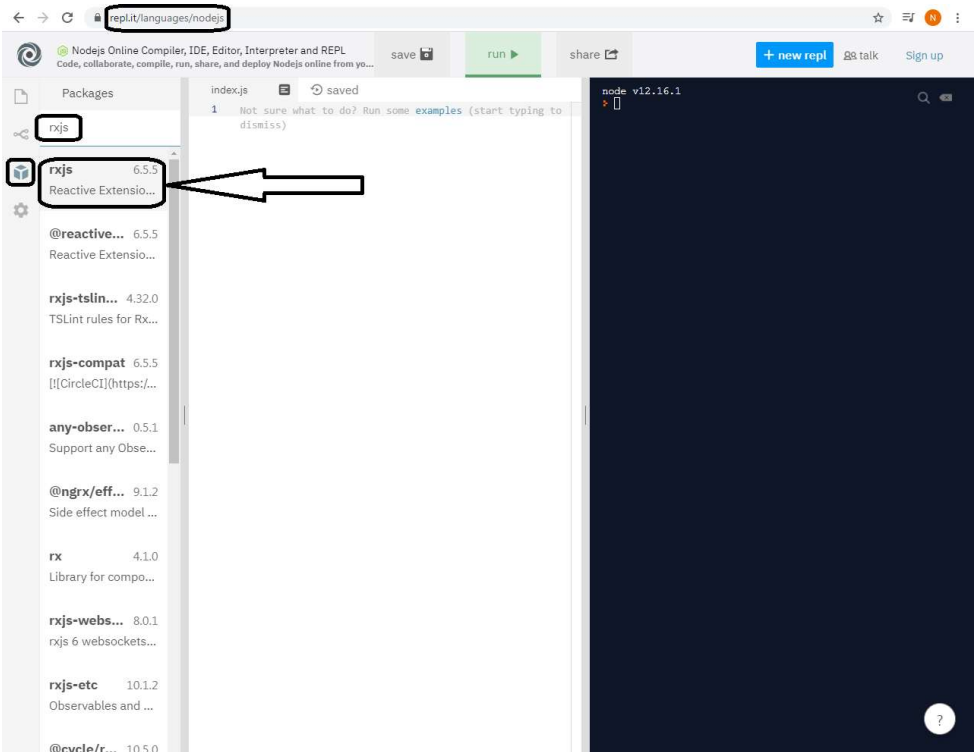

Рис. 1. Пошук пакету RxJS

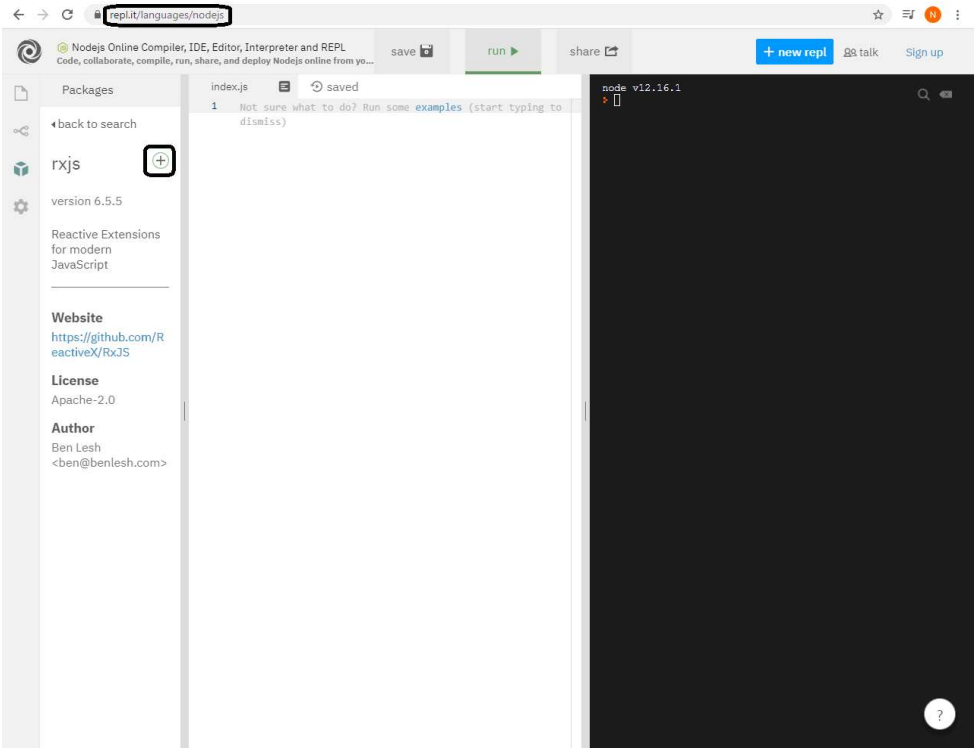

Рис. 2. Додавання пакету RxJS

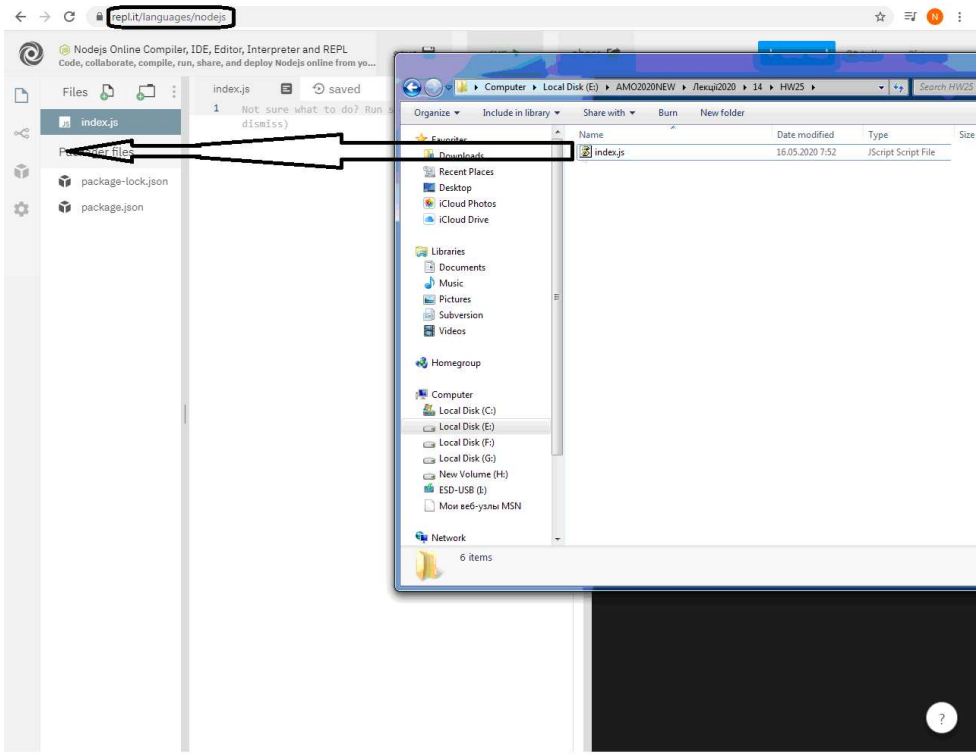

Рис. 3. Перетягування файлу index.js

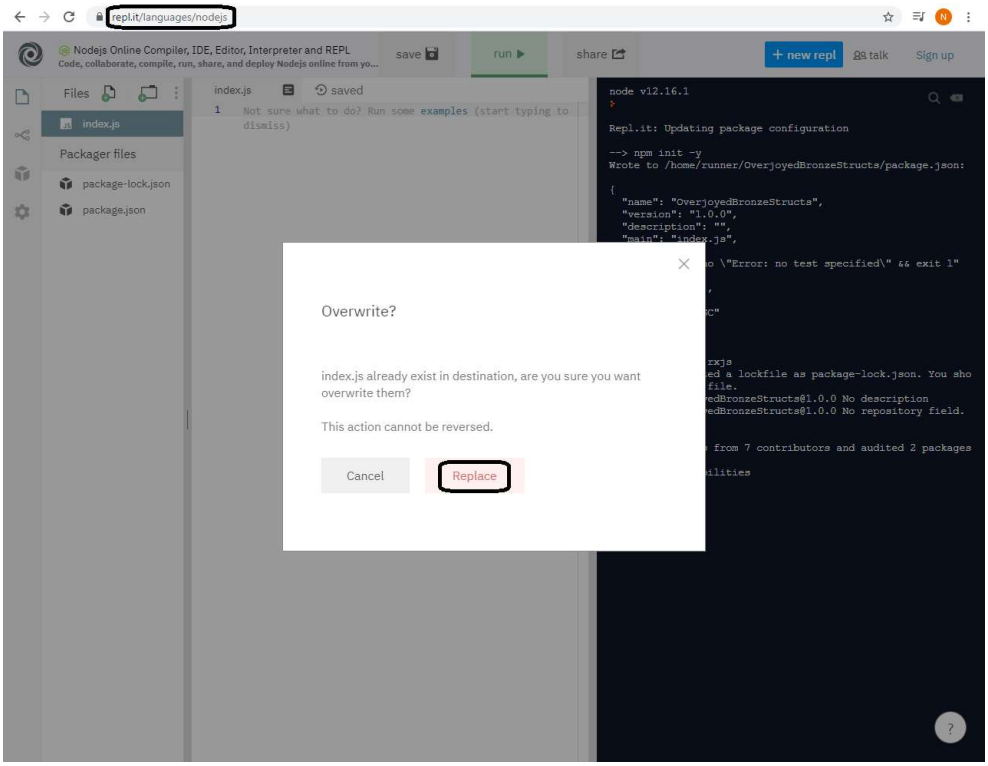

Рис. 4. Підтвердження перезапису файлу index.js

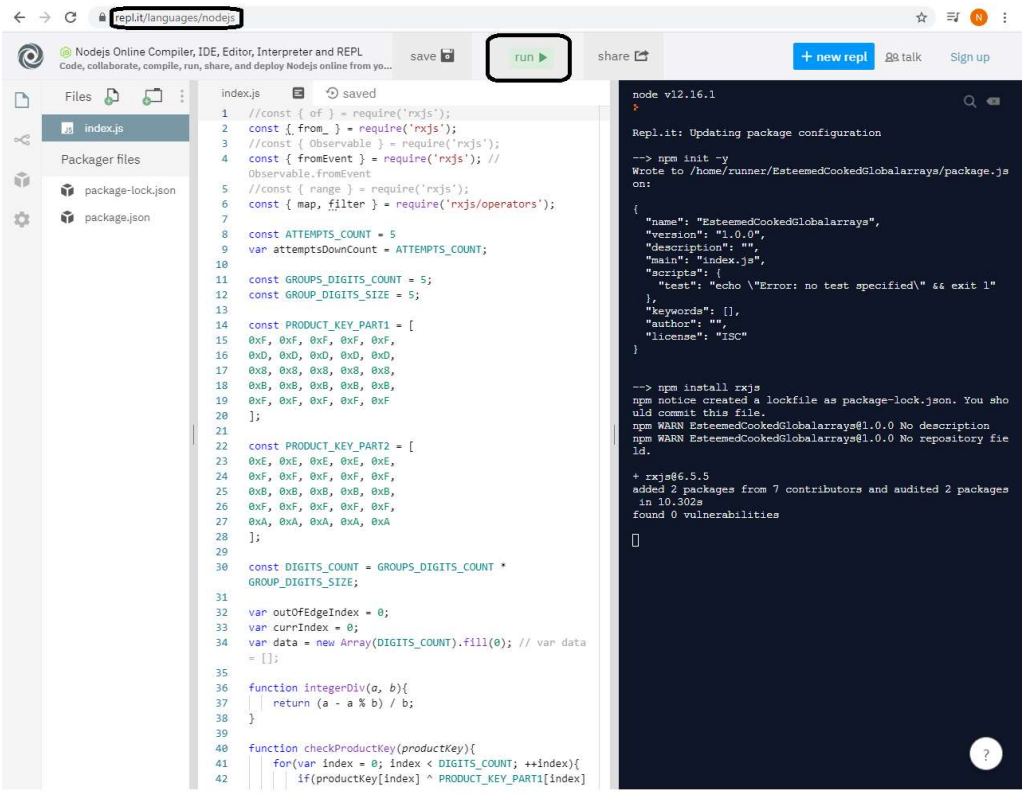

Рис. 5. Запуск програми

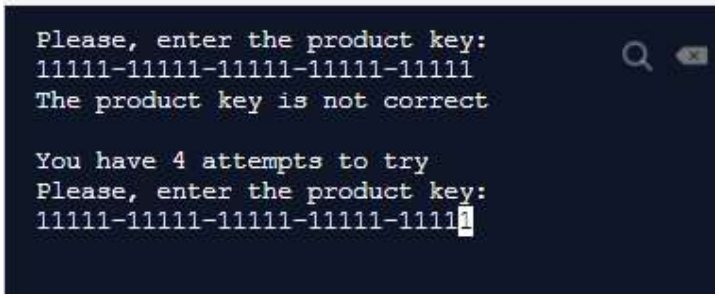

Рис. 6. Повідомлення про помилкове введення ключа ліцензії

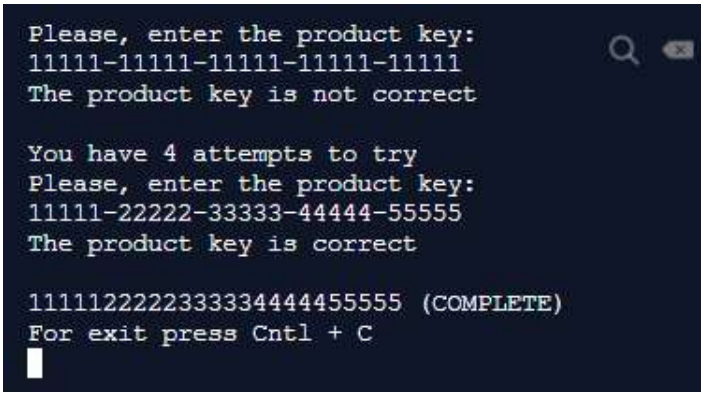

Рис. 7. Повідомлення про коректне введення ключа ліцензії

Приклад коду

** наведений зразок коду реалізовує завдання відповідно до реактивної парадигми програмування за допомогою RxJS, окремо на рушії NodeJS відстежується тільки ввід «Ctrl+C» для завершення роботи програми*

Наведений зразок коду реалізовує завдання для 5-ти максимально допустимих спроб введення ключа ліцензії.

Максимальна кількість спроб для введення ключа ліцензії	5
Оголошення в коді	`const ATTEMPTS_COUNT = 5`

Для коректного виконання коду за допомогою https://repl.it/languages/nodejs віртуальну консоль з правого боку краще трохи розширити перед початком виконання коду, а **у процесі виконання розмір консолі не змінювати**.

Лістинг

```
//const { of } = require('rxjs');
const { from_ } = require('rxjs');
//const { Observable } = require('rxjs');
const { fromEvent } = require('rxjs'); // Observable.fromEvent
const { map, filter } = require('rxjs/operators');

const ATTEMPTS_COUNT = 5
var attemptsDownCount = ATTEMPTS_COUNT;
const GROUPS_DIGITS_COUNT = 5;
const GROUP_DIGITS_SIZE = 5;

const PRODUCT_KEY_PART1 = [
0xF, 0xF, 0xF, 0xF, 0xF,
0xD, 0xD, 0xD, 0xD, 0xD,
0x8, 0x8, 0x8, 0x8, 0x8,
0xB, 0xB, 0xB, 0xB, 0xB,
0xF, 0xF, 0xF, 0xF, 0xF
];

const PRODUCT_KEY_PART2 = [
0xE, 0xE, 0xE, 0xE, 0xE,
0xF, 0xF, 0xF, 0xF, 0xF,
0xB, 0xB, 0xB, 0xB, 0xB,
0xF, 0xF, 0xF, 0xF, 0xF,
0xA, 0xA, 0xA, 0xA, 0xA
];

const DIGITS_COUNT = GROUPS_DIGITS_COUNT * GROUP_DIGITS_SIZE;

var outOfEdgeIndex = 0;
var currIndex = 0;
var data = new Array(DIGITS_COUNT).fill(0); // var data = [];

function integerDiv(a, b){
    return (a - a % b) / b;
}

function checkProductKey(productKey){
    for(var index = 0; index < DIGITS_COUNT; ++index){
        if(productKey[index] ^ PRODUCT_KEY_PART1[index] ^ PRODUCT_KEY_PART2[index]){
            return false;
        }
    }

    return true
```

```
}

function toDigitPosition(currIndex){
    let positionAddon = integerDiv(currIndex, GROUP_DIGITS_SIZE);
    positionAddon && positionAddon >= GROUPS_DIGITS_COUNT ? --positionAddon : 0;
    process.stdout.cursorTo(currIndex + positionAddon);
}

function printProductKey(productKey, outOfEdgeIndex){
    for(var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        process.stdout.write(productKey[index].toString(16) );
    }
}

function printFormattedProductKey(productKey, outOfEdgeIndex){
    for(var index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        process.stdout.write(productKey[index].toString(16) );
        if(!((index + 1) % GROUP_DIGITS_SIZE) && (index + 1) < DIGITS_COUNT){
            process.stdout.write( '-' );
        }
    }
}

function inputHandler(ch, key) {
    if(!attemptsDownCount){
      return;
    }
    if ( key && key.name == 'return' ) {
        if (checkProductKey(data) ) {
            process.stdout.write("\nThe product key is correct\n\n");
            printProductKey(data, outOfEdgeIndex)
            process.stdout.write(' (COMPLETE)');
            process.stdout.write('\nFor exit press Ctrl + C\n');
            attemptsDownCount = 0;
        }
        else{
            process.stdout.write("\nThe product key is not correct\n");
            process.stdout.write("\nYou have " + --attemptsDownCount + " attempts to try");
            if(attemptsDownCount){
                process.stdout.write("\nPlease, enter the product key:\n");
                printFormattedProductKey(data, outOfEdgeIndex);
                toDigitPosition(currIndex);
            }
            else{
                process.stdout.write("\nThe product key is not entered\n");
                process.stdout.write("For exit press Ctrl + C\n");
            }
        }
    }

    if (key && key.name == 'backspace') {
        if(currIndex){
            --currIndex;
            toDigitPosition(currIndex);
            data[currIndex] = 0;
            process.stdout.write( '0' );
            toDigitPosition(currIndex);
        }
    }
    else if (key && key.name == 'delete') {
        toDigitPosition(currIndex);
        data[currIndex] = 0;
        process.stdout.write( '0' );
        toDigitPosition(currIndex);
    }
    else if (key && key.name == 'left') {
        if(currIndex){
            toDigitPosition(--currIndex); // got to 1.5
        }
    }
    else if (key && key.name == 'right') {
```

```
        if(currIndex < outOfEdgeIndex){
            toDigitPosition(++currIndex);
        }
    }

    var hexDigitRegularExpression = /^[0-9A-Fa-f]\b/; // /[0-9A-Fa-f]/g

    if (ch && hexDigitRegularExpression.test(ch) && currIndex < DIGITS_COUNT) {
        data[currIndex] = ch.toUpperCase();
        process.stdout.write( data[currIndex] );
        if(outOfEdgeIndex <= currIndex){
            outOfEdgeIndex = currIndex + 1;
        }
        if(currIndex + 1 < DIGITS_COUNT) {
            ++currIndex;
            if (currIndex != DIGITS_COUNT && !(currIndex % 5)) {
                process.stdout.write( '-' );
            }
        }
        if(currIndex + 1 == DIGITS_COUNT){
            toDigitPosition(currIndex);
        }
    }
}

console.clear();
var keypress = require('keypress');
keypress(process.stdin);
process.stdin.setRawMode(true); // without press enter
process.stdin.setEncoding( 'utf8' );

// Resume stdin in the parent process.
// Node application close all by itself if be an error or process.exit().
process.stdin.resume();

process.stdin.on( 'keypress', (ch, key) => {
    if ( key && key.ctrl && key.name == 'c' ) { // ctrl-c ( return from program )
        process.exit();
    }
});

const inputObservable = fromEvent(process.stdin, 'keypress');

const PrintObserver = {
  next: (key) => {
    inputHandler(key[0], key[1]);
  },
  error: (err) => {
    console.error('something wrong occurred: ' + err);
  },
  complete: () => {
    console.log('done');
  }
};

inputObservable
.pipe(
    //filter is not used
    //filter(key => !(key && key[0] && key[0] == 'Q')),
    map(key =>
    (key && key[0] && (key[0] == ' ' || key[0] == '\t')) ? ['0', key[1]] : key)
)
.subscribe(PrintObserver);

console.clear();
if(attemptsDownCount){
    process.stdout.write('Please, enter the product key:\n');
}
process.stdout.close; // state out fix
```