**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інтелектуальних інформаційних систем**

**ЗВІТ**

**ПРО ТЕХНОЛОГІЧНУ ПРАКТИКУ**

122 – ПРТ.ПЗ.00 – 201. 1820104

Виконав: студент 2 курсу, групи 201,

спеціальності   
122 «Комп’ютерні науки»

\_\_\_\_\_\_\_\_\_\_\_Василенко А.О.

(підпис, ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Керівник: к.т.н., доцент, доцент каф. ІІС

Г.В. Кондратенко

(підпис, ініціали та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**Миколаїв  – 2020 рік**

**ЗМІСТ**

[ВСТУП 3](#__RefHeading___Toc991_31222461)

[1. ЗМІСТОВНА ЧАСТИНА 4](#__RefHeading___Toc993_31222461)

[2. ПРИКЛАДИ ОФОМРЛЕННЯ ЕЛЕМЕНТІВ 5](#__RefHeading___Toc995_31222461)

[ВИСНОВКИ 8](#__RefHeading___Toc997_31222461)

ВСТУП

1) Метою роботи було створення магазину за технологією SPA - single page application. Потрібно було розібрати як працює бібліотека React і її екосистема, також використовувати прийоми оптимізації, зокрема React.memo для компонентів і часткове кешування для бізнес логіки. Знайомство з Flux і Observer паттернами.

Просунута робота з async кодом і композиція елементами магазину.

Навчитися працювати з хмарними базами даних, Firebase в даному проекті.

2) Чи навчиться працювати з TypeScipt, зрозуміти type inference. Метою роботи було створення скрипта для сортування із застосуванням ООП підходів

Викладач - Таранов М.О

ЗМІСТОВНА ЧАСТИНА

1. Магазин за технологією SPA c застосуванням бібліотеки React

Технічний стек:

**React** - бібліотека побудови призначеного для користувача інтерфейсу,

Стиль написання коду в React - декларативний, це дозволяє легко передбачити виконання коду при різних станах.

Самі компоненти є інкапсульованими від інших компонентів, що дає можливість написання reusable components. В основі написання компонентів лежить JSX, який перетворює XAML подібний синтаксис в об'єкти класу, для обробки jsx існує babel і сам webpack через який ми можемо контролювати розмір bundle і робити різні маніпуляції через скачані плагіни і loaders

Для передачі даних існує "component's props" - це

Об'єкт який містить ключ - значення даних які ми передаємо

Через дерево компонентів.

Для маніпулювання даними використовується система станів.

Вона дозволяє раціонально змінювати дані на сторінці, також

Не мало важливим елементів React є lifecycle методи. У свою чергу вони допомагають вести маніпуляцію з даними в різний час, наприклад, якщо нам потрібно зберігати скрол користувача під час переходу на іншу сторінку ми використовуємо "getSnapshotBeforeUpdate"

Або ж додати префікси до форми при написання різного тексту, тоді getDerivedStateFromProps.

Під кришкою React використовує VirtualDom, по суті це те ж саме що і реальним будинок, тільки він заснований на об'єктах наших компонентів і тримається він як не дивно віртуально в пам'яті, що дає повний контроль при роботі движка Fiber. Сам Fiber, по моєму думку, це інновація. Сам движок працює по системі пріоритетів, наприклад, якщо ми візьмемо 2 компонента, перший буде controlled input, а другий списком з n кількістю елементів який щось робить, то Fiber в першу чергу поставить список на паузу, а інпут оновить (нам же потрібен хороший UX при роботі з нашим додатком).

**React-Dom** - обгортка над DOM API,

ReactDOM це інструмент який допомагає зв'язати і використовувати DOM з React. Наприклад, якщо ми можемо уявити React у вигляді робота,

Те ReactDOM - полотно на якому цей робот пише. Головна фішка ReactDom - це кроссплатформенность, яка дозволяє працювати React в старих браузерах. Важливим елементом бібліотеки є перетворення компонента в рядок, що дає можливість використовувати React на стороні сервера. Така бібліотека як Next може це зробити.

**Redux** - бібліотека для state management,

Спочатку React йшов без системи контролю станів, тільки потім, через час, ввели ContextAPI, до цього все використовували props, або сторонні бібліотеки. Сам Redux побудований за Flux паттерну, що дає йому схожість на MVC. Схема така, view компонент звертається до action creator, він в свою чергу дає об'єкт в формі (type, payload) потім цей об'єкт поміщається в dispatch функцію яка направляє це об'єкт в

Redux store, там він проходить через все reducer коли типи підходять ми змінюємо immutable state редьюсера через повернення нового об'єкта. Всі компоненти які були підписані на цей store, отримують оновлення і потім оновлюються. Для спрощення цього процесу використовується HOC компоненти, які можуть обернути певний компонент в потрібний для його функціонал. Для цього що б не ставити підписку і відписку на пряму, ми довіряємо HOC компоненту, в даному випадку connect з react-redux який це все робить за нас. В основі Redux лежить Context API який дає систему Provider Consumer. Так само варто додати, що redux спочатку оптимізований, при змінах стану на той же самий, redux не викликатиме trigger на підключений компонент

**Redux-saga** - бібліотека для обробки асинхронних запитів

(Зокрема використовується в зв'язці з Container патерном),

Саги доповнюють сам Redux. Способом впровадження додаткового шару між srote і view компонентом, ми можемо уявити ситуацію, коли ми хочемо отримати певні дані від запиту, але сам запит нічого не дає, для цього часто використовують цю систему. Вона буквально заморожує відправлений action об'єкт в своєму шарі і робить з ним різні маніпуляції. Якщо ми захочемо, то у випадку помилки ми можемо відправити інший action об'єкт в наш store. Redux-saga побудована на основі функцій генераторів, вони в свою чергу контролюють послідовність виконань в функції (ті ж Проміс і async await є обгорткою над генераторами) ми також можемо мати свій store саг для гнучкості. Звичайно є і альтернативи цій бібліотеці, легші, redux-thunk наприклад, він володіє схожим функціоналом, але в його основі логіка лежить на функціях, якщо раніше у нас був action creator, то тут HOF який замість об'єкта повертає функцію. Middleware в store допомагає відстежити цю функцію, якщо він її знаходить тоді він дає в аргументи dispatch який ми вже використовуємо.

**Styled-components** - бібліотека для інкапсуляції стилів і можливістю використання JS

Ця бібліотека є улюбленою серед любителів інкапсуляції стилів. Зазвичай така бібліотека використовується в зв'язці з UI бібліотеками, також знаходить застосування, коли ми хочемо поміняти стилі об'єкта в залежності від даних в компоненті, наприклад, підсвітити блок при отримання даних або включити іншу анімацію при якомусь дії користувача. Такі бібліотеки входять в групу CSS-in-JS, але крім звичайного css, ми можемо використовувати вбудований предпроцессор SASS

**Redux-persist** - бібліотека для взаємодії з local storage через Provider і enchanced reducer.

Бібліотека яка заснована на white / black list системі. Ми можемо написати будь reducer-и або їх компоненти ми повинні зберігати в storage. При використанні з React ми повинні обернути потрібні нам reducer в enchanced reducer і поставити PersistGate який буде в пропсах приймати persistor. Persistor - це оптимізований під роботу з Redux-persist стор, який ми також обертає в потрібний HOC.

**Reselect** - бібліотека для кешування значень з redux store

Кешування значень стає необхідністю при роботі з великим компонентом у якого багато підключених значень

**Stripe** - система оплати

Stripe виступає посередником між покупцем і продавцем, який проводить безпечний з точки зору захисту інформації платіж. Stripe вирішує всі проблеми платежів картами, включаючи зберігання даних карт, періодичні платежі та висновки на банківський рахунок.

Систему Stripe відрізняє висока швидкість обробки платежів і безпеку, а з точки зору бізнесу - дуже зручні засоби інтеграції з сайтами (особливо для програмістів)

**Firebase** - бібліотека для взаємодії з хмарної базою даних

Серед найкращих і найбільш популярних BaaSs можна виділити Firebase від компанії Google. По суті, Firebase є безумовно приголомшливим у виконанні, реалізації та експлуатації. Firebase служить базою даних, яка змінюється в реальному часі і зберігає дані в JSON. Будь-які зміни в базі даних відразу синхронізуються між усіма клієнтами, або девайсами, які використовують одну і ту ж базу даних. Іншими словами, оновлення в Firebase відбуваються миттєво.

Разом зі сховищем, Firebase також надає призначену для користувача аутентифікацію, і тому всі дані передаються через захищене з'єднання SSL. Ми можемо вибрати будь-яку комбінацію email і пароля для аутентифікації, будь то Facebook, Twitter, GitHub, Google, або щось інше.

В добавку до iOS SDK, у Firebase є SDK для Android і JavaScript. Всі платформи можуть використовувати одну базу даних.

Складно уявити що Firebase з усіма цими функціями бюджетне рішення.

На момент написання цієї статті, в безкоштовному пакеті Firebase може обробляти до 100 одночасних з'єднань. Цього цілком достатньо для роботи популярного додатка. За $ 49 в місяць обмежень на мережеву активність немає.

React-router - бібліотека маршрутизації

React Router це стандартна бібліотека маршрутизації (routing) в React. Він зберігає інтерфейс програми синхронізованим з URL на браузері. React Router справді має користь і широко використовується в додатках React більше з боку Server ніж додатки React з боку Client. Точніше React Router зазвичай використовується в ДОДАТОК React в середовищі NodeJS Server, він дозволяє вам визначати динамічні URL але соответствуюших філософії "Single Page Application" (односторінкового додаток) у React.

2) Розробка скрипта на typescript через ООП методи

Abstract class - це контракт між класами який зобов'язує один клас описати абстрактні поля успадкованого для отримання повного функціоналу класу.

Сomposition - це можливість зберігати посилання на інші класи всередині себе і делегувати методи

TypeScript - це супер версія JavaScript, найпомітніша фішка мови, є type inference. Тим самим, мова чимось починає нагадувати Python, який теж як і typescript є динамічним. Сама мова дуже дружить з ООП. Все Gof і не тільки патерни можна реалізувати без бубна, дуже корисний при роботі з великими проектами на React.

У React ми можемо буквально описати функціонал компонента в верху компонента і при ситуації коли нам захочеться відвідати компонент 2 тижневої давності, ми відразу зможемо зрозуміти за що компонент відповідає за його інтерфейсу. Так само в зв'язці з redux відкриває ще одну булочку типізованих мов, це enum. Іноді нам може бути зручніше описати всі типи action в окремому enum файлі

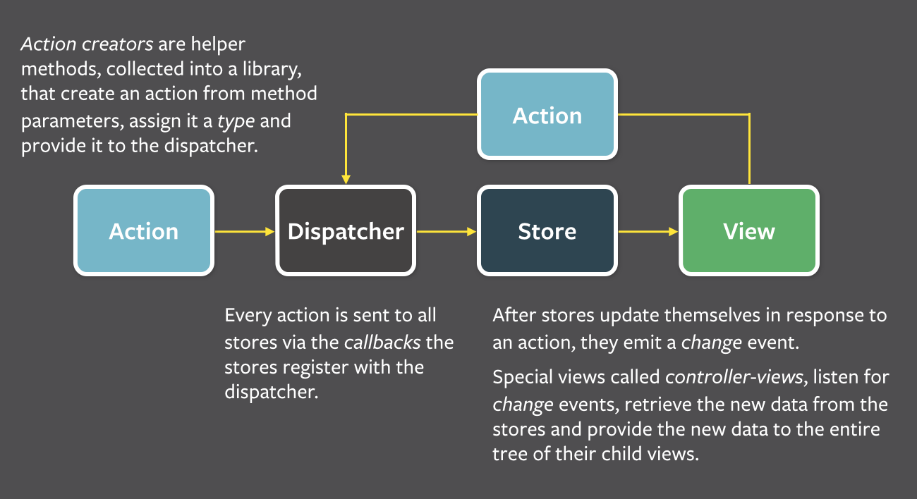


Рис. 1 – Flux pattern

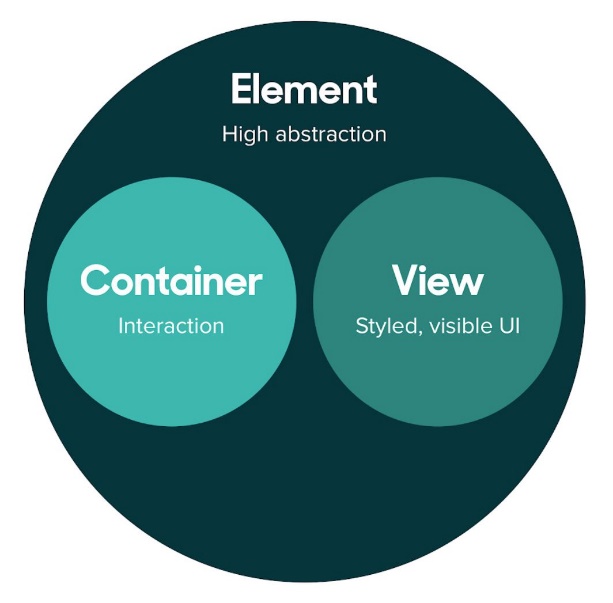


Рис. 2 – Container pattern

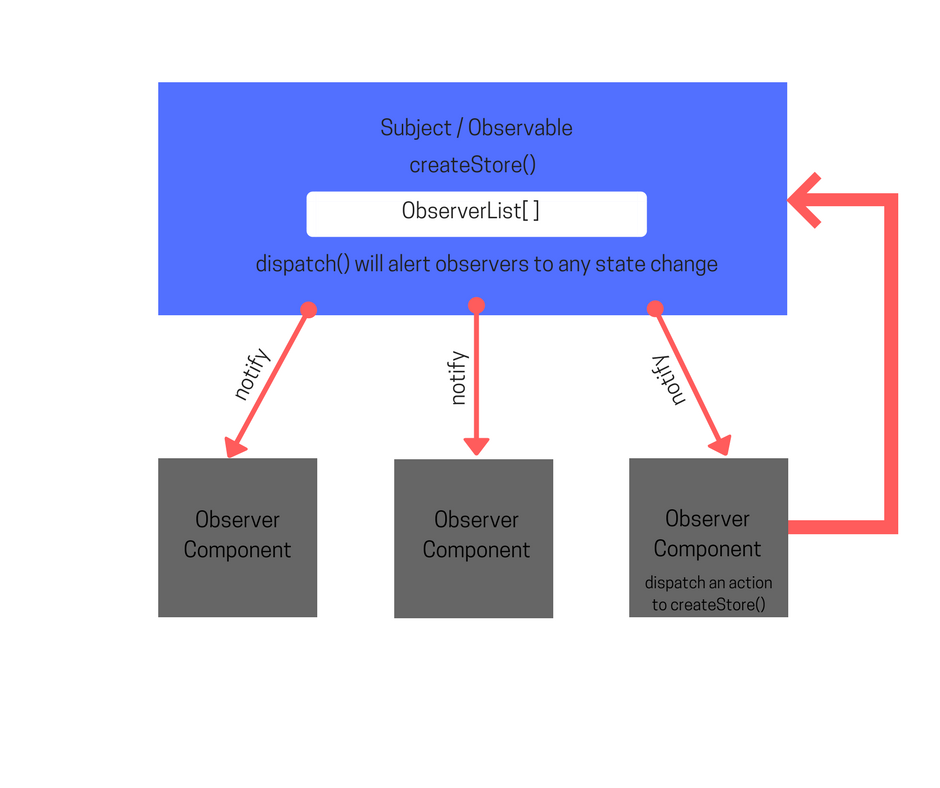


Рис. 3 – Observer pattern

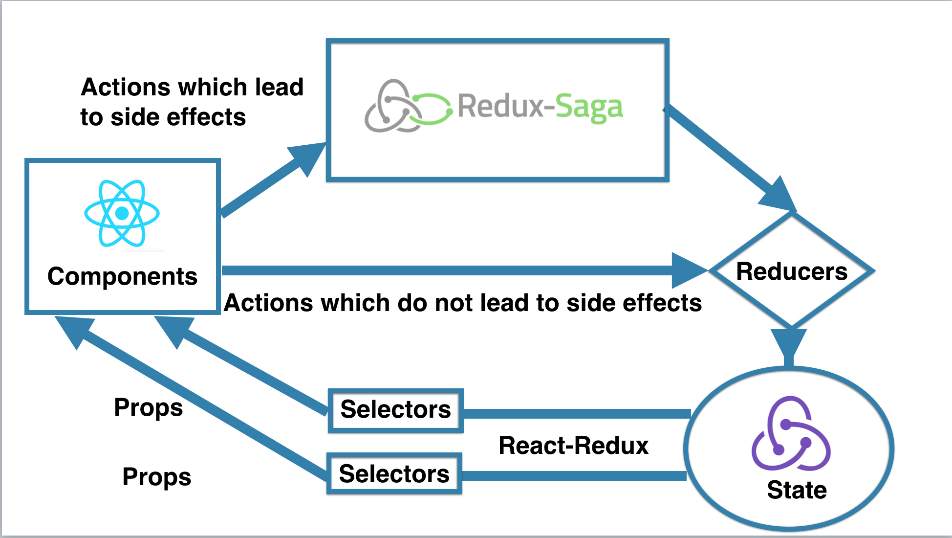


Рис. 4 – Redux-saga flow

1. ПРИКЛАДИ ОФОМРЛЕННЯ ЕЛЕМЕНТІВ

1)SPA магазин на React + Redux

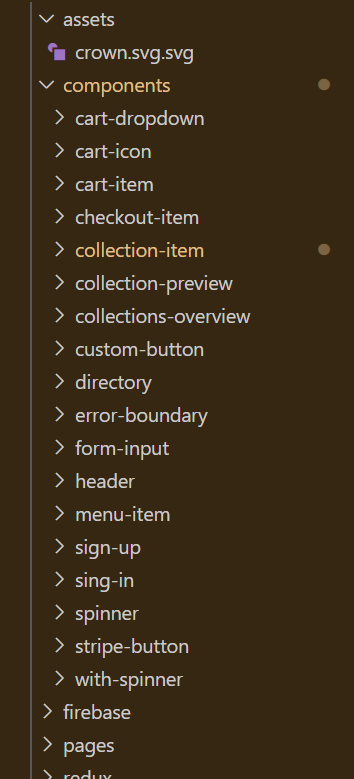


Рис. 5 – Дерево файлов

Детальніше - <https://github.com/RoastDonald/clothing-shop>

2) Алгоритм сортування із застосуванням TypeScript

**Index.ts**

import { Sorter } from './Sorter';

import { NumbersCollection } from './NumbersCollection';

import { CharactersCollection } from './CharactersCollection';

import { LinkedList } from './LinkedList';

const characters = new CharactersCollection('tetAasTt');

characters.sort();

export abstract class Sorter {

  abstract compare(left: number, right: number): boolean;

  abstract swap(left: number, right: number): void;

  abstract length: number;

  sort(): void {

    const length = this.length;

    for (let i = 0; i < length; i++) {

      for (let j = 0; j < length - i - 1; j++) {

        if (this.compare(j, j + 1)) {

          this.swap(j, j + 1);

        }

      }

    }

  }

}

import { Sorter } from './Sorter';

export class NumbersCollection extends Sorter {

  constructor(public data: number[]) {

    super();

  }

  get length(): number {

    return this.data.length;

  }

  compare(leftIndex: number, rightIndex: number): boolean {

    return this.data[leftIndex] > this.data[rightIndex];

  }

  swap(leftIndex: number, rightIndex: number): void {

    let temp = this.data[leftIndex];

    this.data[leftIndex] = this.data[rightIndex];

    this.data[rightIndex] = temp;

  }

}

**LinkedList.ts**

import { Sorter } from './Sorter';

class Node {

  next: Node | null = null;

  constructor(public value: number) {}

}

export class LinkedList extends Sorter {

  head: Node | null = null;

  add(data: number): void {

    let node = new Node(data);

    if (!this.head) {

      this.head = node;

      return;

    }

    let tail = this.head;

    while (tail.next) {

      tail = tail.next;

    }

    tail.next = node;

  }

  get length(): number {

    if (!this.head) {

      return 0;

    }

    let tail = this.head;

    let len = 1;

    while (tail.next) {

      len++;

      tail = tail.next;

    }

    return len;

  }

  at(index: number): Node {

    if (!this.head) {

      throw new Error('Index out of the boundry');

    }

    let counter = 0;

    let node: Node | null = this.head;

    while (node) {

      if (counter === index) {

        return node;

      }

      counter++;

      node = node.next;

    }

    throw new Error('Index out of the boundry');

  }

  compare(left: number, right: number): boolean {

    if (!this.head) {

      throw new Error('bb');

    }

    return this.at(left).value > this.at(right).value;

  }

  swap(left: number, right: number): void {

    let leftNode = this.at(left);

    let rightNode = this.at(right);

    let temp = leftNode.value;

    leftNode.value = rightNode.value;

    rightNode.value = temp;

  }

  print(): void {

    if (!this.head) {

      console.log('empty');

      return;

    }

    let node: Node | null = this.head;

    while (node) {

      console.log(node.value);

      node = node.next;

    }

  }

}

**CharactersCollection.ts**

import { Sorter } from './Sorter';

export class CharactersCollection extends Sorter {

  constructor(public data: string) {

    super();

  }

  get length(): number {

    return this.data.length;

  }

  compare(left: number, right: number): boolean {

    return (

      this.data[left].toLocaleLowerCase() > this.data[right].toLocaleLowerCase()

    );

  }

  swap(left: number, right: number): void {

    let newString = this.data.split('');

    let temp = newString[left];

    newString[left] = newString[right];

    newString[right] = temp;

    this.data = newString.join('');

  }

}

ВИСНОВКИ

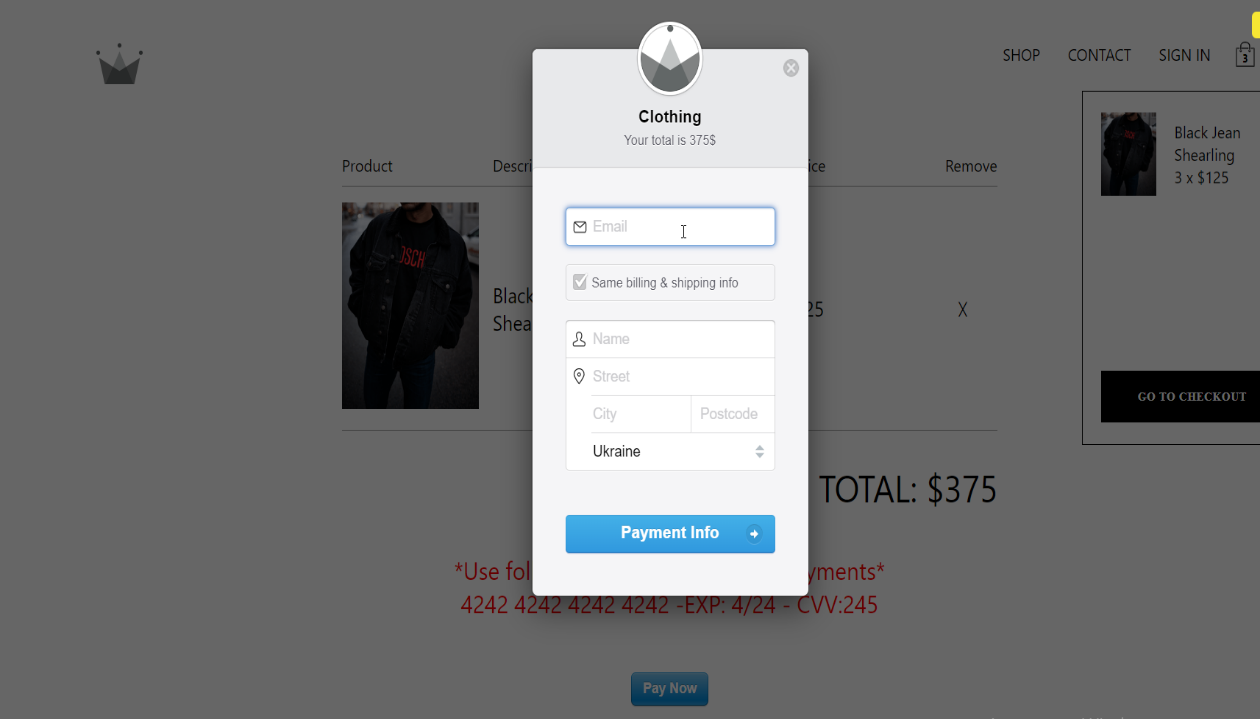


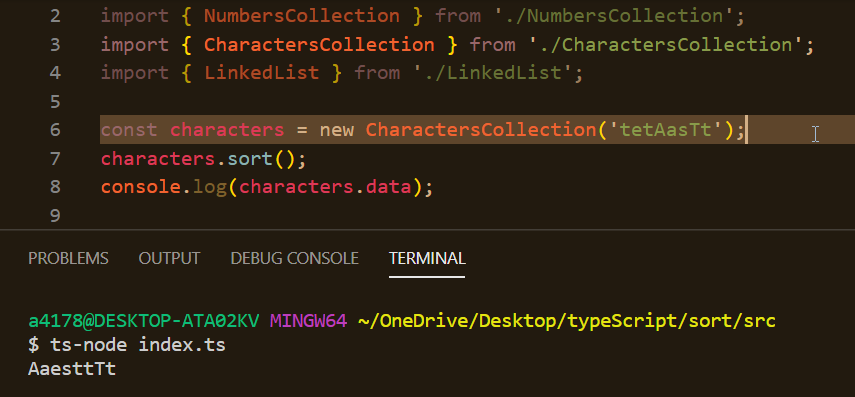
Рис. 6 – Результат роботи №1 

Рис. 7 – Результат роботи №2