Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

факультет інформаційних технологій
(факультет)

Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
Кваліфікаційної роботи ступеня
бакалавра
(назва освітньо-кваліфікаційного рівня)

Студента Ткаченко Ілля Андрійович
(ПІБ)
академічної групи 126-17-1
(шифр)
спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)
за освітньо-професійною програмою
«Інформаційні системи та технології»
(офіційна назва)
на тему «Розробка SRM system»

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
Кваліфікаційної роботи	Гнатушенко В.В.			
розділів:	4			

Рецензент			

Нормоконтроль	Гнатушенко В.В.		

Дніпро
2021

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
___інформаційних технологій та комп'ютерної інженерії
(повна назва)
_____
(підпис) (прізвище, ініціали)

«_____»_____20__ року

ЗАВДАННЯ

на диплому роботу ступеня

_____бакалавра_____
(бакалавра, спеціаліста, магістра)

студенту _Ткаченко І.А._____ академічної групи_____
(прізвище та ініціали) (шифр)
спеціальності___126 «Інформаційні системи та технології»_____
за освітньою-професійною програмою_____
_____«Інформаційні системи та технології»_____
на тему__«Розробка SRM system»_____

Розділ	Зміст	Термін виконання
Розділ 1	Опис тенхологій, які використовуються в дипломній роботі	
Розділ 2	Чому варто використовувати typescript у сучасних веб-програмах	
Розділ 3	PWA і SPA програми та їхні особливості	
Розділ 3	Опис веб додатка	

Завдання видано _____ ___В.В. Гнатушенко_____
(підпис керівника) (прізвище,ініціали)

Дата видачі завдання: _____

Дата подання до екзаменаційної комісії _____

Завдання прийняв до виконання _____ _Ткаченко І.А._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 62 с., 28 рис 12 джерело.

Об’єкт розробки: CRM система для впорядкування клієнтів і продуктів компанії, і для ведення обліку.

Мета роботи: Спрощення ведення обліку та взаємодії між клієнтами, і огляд нової технології typescript та Створення веб-додаток SPA

Предмет дослідження: Створення веб додатки та вивчення можливостей мови typescript і особливостей SPA

Практичне значення кваліфікаційної роботи полягає в тому що б спростити роботу менеджерів з клієнтами і продуктами які вони замовили

Розроблене технологічне рішення може бути використано для взаємодії менеджера з клієнтами та замовленими продуктами, дане рішення підійде для малого і середнього бізнесу

CRM, SPA, TYPESCRIPT, ВЕБ-ДОДАТОК

ABSTRACT

Explains the note: 62 p., 28 rice 12 link.

Development report: CRM system for ordering customers and products of the company, and for conducting business.

Meta robots: Promoting the maintenance of the region and interaction between the clients, and looking at the new technologies of typescript and the creation of the SPA web add-ons

Subject of advancement: Web add-ons and features of mov typescript and SPA features

The practical importance of high-quality robots in order to help the robot managers with customers and products have been replaced

The development of technological solutions can be used for the manager's interaction with customers and replacement products, given a solution for small and middle business

CRM, SPA, TYPESCRIPT, WEB-APPLICATION

ЗМІСТ

ВСТУП

CRM Абревіатура - акронім англійського словосполучення "Customer Relationship Management", що російською звучить як "Управління відносинами з клієнтами".

Простіше кажучи, CRM - система управління взаємовідносини з клієнтами, за допомогою якої можна автоматизувати рутинні бізнес-процеси компанії і налагодити ефективну роботу всередині компанії.

Клієнтська база на паперових носіях (всілякі картотеки, замітки в блокнотах, товстих зошитах тощо) - минуле століття. Електронні органайзери і популярний Excel так само не забезпечують ефективне взаємовідношення між потенційним клієнтом і постачальником товарів, послуг. Основна причина неефективності - банальний людський фактор: який-небудь із співробітників компанії обов 'язково забуде внести інформацію в файл Excel, вважатиме її малозначною або допустить помилку.

Ситуація обростає проблемами в геометричній прогресії, коли зростають продажі, що, природно, впливає на розширення штату співробітників, збільшення кількості клієнтів. Розв'язання проблеми - CRM-система, яка автоматизує бізнес-процеси і, як наслідок, ліквідує помилки, викликані людським фактором.

Будь-який бізнесмен рано чи пізно приходить до того, що бізнес начебто працює і все досить непогано, але є величезне кол-во вад, вирішивши які можна вийти зовсім на інший рівень бізнесу і менеджменту.

Щоб перейти на новий етап, потрібно розібратися з нагальними проблемами і рутиною, впровадивши CRM.

Основна суть цієї роботи це спрощення роботи і ведення клієнтів, і інформацію про продаж і про сам продукт.

CRM система розроблена на фреймірвці vue.js.

Мова, яку було використано для написання typescript.

РОЗДІЛ 1

Опис тенхологій, які використовуються в дипломній роботі

1.1 Axios

1.1.1 Порівняння Axios та Fetch

Axios - це JavaScript-бібліотека для виконання або HTTP-запитів у Node.js, або XMLHttpRequests у браузері. Вона підтримує проміси - новинку ES6. Одна з особливостей, яка робить її кращою за fetch () - автоматичні перетворення JSON-даних.

При використанні fetch () для передачі даних в JSON, необхідно виконати процес у два етапи. Спочатку зробити фактичний запит, а потім викликати метод json () для отриманих даних з сервера. Коли бібліотека Axios стала популярною, в браузерах не було API, що реалізує HTTP-клієнт, заснований на промісах.

Стандартний інтерфейс XML HTTP Request (XHR) був незручним, працювати з ним було важко. Розробники з радістю прийняли Axios через те, що ця бібліотека полегшувала їм життя.

У 2015 вийшов API Fetch. Чому ж ми, в 2019 році, досі використовуємо Axios? Порівняймо ці дві технології.

Fetch: рис 1.1

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
// {
//   "userId": 1,
//   "id": 1,
//   "title": "delectus aut autem",
//   "completed": false
// }
```

Рис 1.1 Приклад використання fetch

Axios: рис 1.2

```
axios.get("https://jsonplaceholder.typicode.com/todos/1")
  .then(response => console.log("response", response.data))
// {
//   "userId": 1,
//   "id": 1,
//   "title": "delectus aut autem",
//   "completed": false
// }
```

Рис 1.2 Приклад використання axios

При використанні Fetch доводиться мати справу з двома промісами. А ось при роботі з Axios у нас є прямий доступ до JSON-результату у властивості data об 'єкта відповіді.

Метод json () міксину Body приймає потік Response і повністю читає його. Він повертає проміс, який дозволяється JSON-результатом розбору тексту тіла запиту.

Ще більше шаблонного коду в Fetch доводиться використовувати при роботі з POST-запитами.

Fetch: 1.3 рис

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST",
  body: JSON.stringify({
    title: "Title of post",
    body: "Post Body"
  })
})
  .then(res => {
    if (!response.ok) throw Error(response.statusText);
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

Рис 1.3 Приклад запиту fetch

Axios: рис 1.4

```
axios
  .post("https://jsonplaceholder.typicode.com/posts", {
    title: "Title of post",
    body: "Body of post"
  })
  .then(response => console.log(response.data))
  .catch(error => console.log(error));
```

Рис 1.4 Приклад запиту axios

Використання Axios дозволяє уникнути написання великих обсягів шаблонного коду і зробити код чистішим і зрозумілішим

1.1.2 Обробка помилок

Fetch: рис 1.5

```
fetch("https://jsonplaceholder.typicode.com/todos/100000")
  .then(response => {
    if (!response.ok) throw Error(response.statusText);
    return response.json();
  })
  .then(data => console.log("data", data))
  .catch(error => {
    console.log("error", error);
  });
// error Error: Not Found
```

Рис 1.5 Приклад обробки помилок fetch

Axios:

```
axios
  .get("https://jsonplaceholder.typicode.com/todos/100000")
  .then(response => {
    console.log("response", response);
  })
  .catch(error => {
    console.log("error", error);
  });
// error Error: Not Found
```

Рис 1.6 Приклад обробки помилок axios

Бібліотека Axios видає інформацію про мережеві помилки, а API Fetch – ні. Працюючи з Fetch завжди потрібно перевіряти властивість response.ok. Для того щоб спростити рішення даного завдання, перевірку цієї помилки можна оформити у вигляді окремої функції: рис 1.7

```
const checkForError = response => {
  if (!response.ok) throw Error(response.statusText);
  return response.json();
};
fetch("https://jsonplaceholder.typicode.com/todos/100000")
  .then(checkForError)
  .then(data => console.log("data", data))
  .catch(error => {
    console.log("error", error);
  });
```

Рис 1.7 Приклад функції axios

1.2 Chart.js

Chart. js - це популярний інструмент, який призначений для створення графіків і діаграм. Ви зможете створювати адаптивні діаграми будь-якої складності на основі HTML5 Canvas

Ця бібліотека дозволяє без особливих зусиль створювати графіки та діаграми будь-якого типу, а також вибудовувати дані на діапазоні часу та логарифмічній шкалі. Також в неї вбудовані засоби роботи з анімацією, що дозволить ефектно видозмінювати графіки залежно від нових даних, а також експериментувати з кольором

Ви можете завантажити останню версію Chart.js з GitHub або з 'єднати бібліотеку до вашого проекту, використовуючи CDN посилання. Крім того, можна налаштувати npm або bower, виконавши одну з наступних команд.

```
npm install chart.js --save
bower install chart.js –save
```

Давайте представимо таблицю з чисельністю населення країн світу у вигляді стовпчої діаграми. На осі у відобразимо цифри, а на осі x назва країн. Для початку створимо елемент canvas з ідентифікатором popChart:

```
<canvas id="popChart" width="600" height="400"></canvas>
```

Атрибути width і height визначають розміри діаграми. Для того щоб графік був адаптивним ми повинні визначити ширину і висоту елемента canvas.

Далі нам необхідно ініціалізувати клас Chart. Зробити це можна, вибравши елемент сторінки, ініціювавши бібліотеку через jQuery або 2d контекст елемента canvas.

```
var popCanvas = $("#popChart");
var popCanvas = document.getElementById("popChart");
var popCanvas = document.getElementById("popChart").getContext("2d");
var barChart = new Chart(popCanvas, {
  type: 'bar',
  data: {
```

```
  labels: ["China", "India", "United States", "Indonesia", "Brazil", "Pakistan", "Nigeria", "Bangladesh", "Russia", "Japan"],
  datasets: [{
    label: 'Population',
    data: [1379302771, 1281935911, 326625791, 260580739, 207353391, 204924861, 190632261, 157826578, 142257519, 126451398],
    backgroundColor: [
      'rgba(255, 99, 132, 0.6)',
      'rgba(54, 162, 235, 0.6)',
      'rgba(255, 206, 86, 0.6)',
      'rgba(75, 192, 192, 0.6)',
      'rgba(153, 102, 255, 0.6)',
      'rgba(255, 159, 64, 0.6)',
      'rgba(255, 99, 132, 0.6)',
      'rgba(54, 162, 235, 0.6)',
      'rgba(255, 206, 86, 0.6)',
      'rgba(75, 192, 192, 0.6)',
      'rgba(153, 102, 255, 0.6)'
    ]
  }]
 }
});
```

Chart.js витягне інформацію з об 'єкта який ми зрадили і намалює відповідний графік. У параметрі type необхідно вказати тип діаграми. Це поле може приймати одне з наступних значень: line, bar, radar, polarArea, pie, doughnut или bubble. У даній серії статей ми розглянемо окремо кожен вид.

Самі дані передаємо через ключ data. У параметрі backgroundColor можна вказати колір, до якого пофарбуються стовпчики діаграми. Типовим значенням є "rgba (0,0,0,0.1)".

Насправді для кожного типу діаграм характерні особливі характеристики. В результаті ми отримаємо наступний результат: Рис 1.8

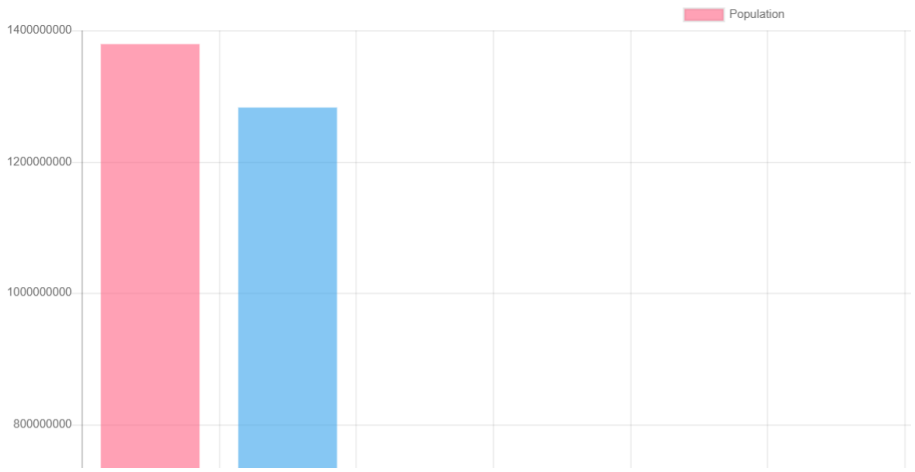

Рис 1.8 результат виведення

1.3 Core-js

Поліфіл або Поліфілл у веб-програмуванні це код, що реалізує будь-яку функціональність, яка не підтримується в деяких версіях веб-браузерів. Зазвичай бібліотеку JavaScript, яка реалізує підтримку веб-стандарту HTML5 у версіях переглядачів, де підтримка цих функцій частково або повністю відсутня.

Це забезпечує більш-менш однакове відображення веб-сторінок у різних веб-переглядачах.

Core-js це найбільш популярний і потужний поліфіл стандартної бібліотеки JavaScript. Включає поліфіли можливостей мови за стандартом ECMAScript аж до 2019: обіцянки, символи, колекції, ітератори, типізовані масиви, багато інших можливостей, пропозиції ECMAScript, деякі крос-платформенні можливості зі стандартів і пропозицій WHATWG/ W3C, на кшталт URL.

Ви можете завантажувати тільки потрібні вам модулі або використовувати поліфіл без забруднення глобального простору назв.

Інтегрований з Babel, що дозволяє автоматично додавати необхідні додатки core-js до вашого коду.

1.4 register-service-worker

register () метод ServceWorkerContainer інтерфейсу, який створює і оновлює ServiceWorkerRegistration для вказаного URL js скрипту.

Якщо вдало, service worker registration зв 'язується за вказаною URL js скрипту, який відповідно використовується для перевірки збігу при навігації за URL. Якщо метод не повертає ServceWorkerRegistration, він повертає Promise. Є можливість викликати цей метод без перевірки на умову, тобто немає необхідності спочатку перевіряти чи існує активна реєстрація в даний момент чи ні

```
ServiceWorkerContainer.register(scriptURL, options)
  .then(function(ServiceWorkerRegistration) { ... });
```

1.5 Vue

Vue (вимовляється/vju^/, приблизно як view) - це прогресивний фреймворк для створення інтерфейсів користувача. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами і додатковими бібліотеками.

Це JavaScript-фреймворк з відкритим вихідним кодом для створення інтерфейсів користувача.

На даний момент підтримується творцем Еваном Ю. і іншими активними членами основної команди з різних компаній, таких як Netlify, Netguru, Baidu, Livestorm.

Опитування, проведене в 2016 році для JavaScript, показало, що Vue має 89% задоволеності розробників. На GitHub проект заробляє в середньому 95 зірок є третім за величиною проектом в історії Github.

Розробники називають Vue.js прогресивним і поступово адаптованим порівняно з іншими веб-фреймворками.

Це дозволяє розробнику налаштувати структуру програми відповідно до власних вимог. Розробники вважають Vue.js більш простим в освоєнні, ніж AngularJS, оскільки API побудований набагато простіше в освоєнні. У Vue.js можна використовувати тільки знання JavaScript і HTML. Можна застосувати Typescript. У Vue.js є власна офіційна досить багата документація на багатьох мовах, викладена на vuejs.org , яка може послужити прикладом у поясненні проектування і розробки в браузері.

У Vue.js реалізується шаблон MVVM, Vue.js пропонує можливість прив 'язки даних на Javascript, так що висновок і введення даних пов' язуються безпосередньо з джерелом даних. Таким чином, режим ручного визначення даних (наприклад, через jQuery) з HTML-DOM не потрібен. При цьому немає необхідності в ніяких додаткових анотаціях, як в Knockout.js, оголошені в Vue-Element звичайні змінні JavaScript включаються в якості реактивних елементів.

1.6 Концепції Vue.js

Основними концепціями Vue є:

1. **Конструктор**

У першій версії ще були фільтри, але, наскільки я знаю, в поточній версії вони вважаються deprecated.

Робота з Vue.js починається зі створення нового інстансу new Vue. В el у нас елемент, за яким стежить Vue. У template вибрано (або прописано інлайнове) елемент, куди Vue буде рендерувати. У data зберігається поточний стан інстансу, а метод computed надає нам вираховувані властивості.

У прикладі обчислювана властивість full_name відстежує first_name і last_name як залежності і автоматично синхронізується.

У methods можна виділити такі кастомні методи і методи життєвого циклу Vue:

- beforeCreate - дивиться дані та ініціалізує події;
- created - дивиться, чи є el або template. Якщо є, то рендерить у них; якщо ні, то шукає метод render;
- beforeMount - створює vm. $ el і замінює el;
- mounted - елемент відреендерен.

При зміні стану:

- beforeUpdate - знову рендерить VDOM і порівнює з реальним DOM-ом, застосовує зміни;
- updated - зміни відреендерени;
- beforeDestroy - повний демонтаж вотчерів, внутрішніх компонентів і слухачів подій;
- destroyed - викликається, коли виконання дії зупиняється.

```
new Vue({
   el: '<jQueryStyleSelector>',
   template: '<id || inline template>',
   data: {
      props: 'Это видно в шаблонах',
      first_name: "Вася",
```

```
        last_name: "Пупкин"
    },
    computed: {
        full_name: function(){
            return this.first_name + this.last_name; //Вася Пупкин
        }
    },
    methods: {
        // методы жизненного цикла
        beforeCreate: function(){},
        created: function(){},
        beforeMount: function(){},
        mounted: function(){},
        beforeUpdate: function(){},
        updated: function(){},
        beforeDestroy: function(){},
        destroyed: function(){},

        customMethodsAlso: function(){
            //здесь у нас тоже есть доступ к data
        }
    }
})
```

2. Директиви

Директиви - спеціальні атрибути для додавання елементам html додаткової функціональності.

Розглянемо деякі вбудовані директиви (хто працював з Angular, тому вони здадуться дуже знайомими):

- V-bind - динамічно пов 'язується з одним або кількома атрибутами.
- V-cloak - ховає "вусаті" вирази, поки не підтягнулися дані
- V-if - умова для рендера елемента
- V-else - означає "else блок" для v-if
- V-for - циклічно проходить масив об 'єктів
- V-model - пов 'язує стан з input елементом
- V-on - пов 'язує слухача події з елементом
- V-once - рендерить елемент тільки спочатку і більше не стежить за ним
- V-pre - не компілює елемент і його дочірні елементи
- V-show - перемикає видимість елемента, змінюючи властивість CSS display
- V-text - оновлює textContent елемента

Всі Vue-директиви мають префікс "v-". У директиву передається якесь значення стану, а як аргументи можуть бути атрибути html або події.

```
<div v-my-directive="someValue"></div>
Vue.directive('my-directive', {
    bind: function () {
        //підготовчі роботи
        //додавання слухачів подій та інших ресурсомістких функцій,
        //які слід запустити лише один раз
    },
    update: function (newValue, oldValue) {
        /робимо щось з оновленим значенням
    },
    unbind: function () {
        //Очищаємо
        //видаляємо слухачів подій, доданих до bind ()
    }
```

```
})
```

3. Компоненти

Компоненти допомагають розширити основні html елементи і впровадити переіспользований код. По суті, компоненти - повторно використовувані частини UI. На етапі проектування ми розбиваємо наш додаток на незалежні частини і отримуємо деревовидну структуру компонентів.

У Vue.js немає особливих вимог до імен компонентів, але хороша практика - дотримуватися правил W3C щодо кастомних компонентів, тобто букви нижнього регістру і поділу через дефіс.

```
Vue.component('simple-counter', {
  template: '<button v-on:click="counter += 1">{{ counter }}</button>',
  data: function () {
    return {
      counter: 0
    }
  }
})
new Vue({
  el: '#demo'
})
```

Комунікація між vue-компонентами здійснюється за принципом "Props in, Events out". Тобто від батьківського елемента до дочірнього інформація передається через пропси, а назад - викликаються події.

Також у Vue.js є так звані однофайлові компоненти. Ми створюємо файл з розширенням .vue і пишемо туди стилі, шаблон і логіку. Причому писати

можна на будь-якому зручному вам препроцесорі (SASS, Stylus, PostCSS, Jade,...) і мові, що компілюється в JS (CoffeeScript, TypeScript).

```
<style lang="sass">
button {
   border: 1px solid gray;
   &.blue { border-color: blue; }
}
</style>
<template lang="jade">
avatar(:user='user')
input(type='text', v-model='content')
button.blue(@click='submitComment')
</template>

<script>
import Comment from '../models'
import avatar  from './components/avatar.vue'
export default {
   props: ['user'],
   components: {
      avatar
   },
   data () {
      return {
         content: ''
      }
   },
   methods: {
      submitComment (e) {
```

```
            e.preventDefault();
            var comment = new Comment(this.content)
            comment.save().then(() => {
               alert('o_O')
               this.content = ''
            })
         }
      }
   }
</script>
```

4. Перехід

Vue надає різні способи застосування анімаційних ефектів, коли елементи відображаються, оновлені або видаляються з DOM. Вони включають в себе інструменти для:

- автоматичне застосування класів для переходів CSS та анімацій
- інтеграції сторонніх бібліотек для CSS-анімацій, таких як Animate.css
- використання JavaScript для маніпуляції DOM-ом
- інтеграції сторонніх JavaScript бібліотек для анімацій, таких як Velocity.js

Розгляньмо простий приклад:

```
<div id="demo">
  <button @click="show = !show">Toggle show</button>
  <transition name="bounce">
    <p v-if="show">Look at me!</p>
  </transition>
</div>
```

JS

```
new Vue({
  el: '#demo',
  data: {
    show: true
  }})
```

1.6.1 Екосистема фреймворку

1.6.1.1 Роутінг

У Vue.js за маршрутизацію відповідає окремий пакет vue-router. Він підтримує вкладені маршрути до вкладених компонентів, пропонує спрощене API для навігаційних хуків, керовану поведінку скролу і просунутий контроль переходів.

app.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import App from './app.vue'
import ViewA from './view-a.vue'
import ViewB from './view-b.vue'

Vue.use(VueRouter)
const router = new VueRouter()
router.map({
  '/a': { component: ViewA },
  '/b': { component: ViewB }
})
router.start(App, '#app')
```

app.vue

```
<div>
  <h1> Це шаблон, який не буде змінюватися</h1>
  <router-view><!-- вибрані компоненти змінюються --></router-view>
```

```
</div>
```

1.6.1.2 Ajax-запити

Для роботи з Ajax-запитами існує плагін vue-resource. Він надає можливості для створення веб-запитів та обробки відповідей за допомогою XMLHttpRequest або JSONP. Також особливістю плагіну є підтримка Promise API і URI шаблонів.

```
{
  // GET /someUrl
  this.$http.get('/someUrl').then((response) => {
    // успех
  }, (response) => {
    // или ошибка
  });
}
```

1.6.1.3 Керування станом

Vuex - патерн і бібліотека управління станом для додатків на Vue.js. Він надає централізований загальний стан для всіх компонентів у додатку і правила, що забезпечують передбачувану зміну стану.

На зображенні нижче представлено додаток на Vue + Vuex з наступними частинами:

- Стан (State), який є єдиним джерелом даних для компонентів.
- Vue-компоненти (Vue-components), які по суті є лише декларативним відображенням статків.
- Дії (Actions), які відловлюють подію, що сталася, збирають дані з зовнішніх API і запускають потрібні мутації.
- Мутації (Mutations) - єдина частина, яка може змінювати стан і, отримавши дані від Actions, застосовує їх на стані.

1.6.2 vue-class-component

Vue-class-component скорочує процес розробки, дозволяючи розробникам додавати властивості даних і обробники прямо як властивості для класу.

Компонент класу Vue - це бібліотека, що дозволяє створювати компоненти Vue в синтаксисі стилю класу. Наприклад, нижче наведено простий компонент лічильника, написаний за допомогою компонента класу Vue.

Як показано у прикладі, можна визначити дані і методи компонента в інтуїтивно зрозумілому і стандартному синтаксисі класу шляхом анотування класу за допомогою @ Component decorator. Можна просто замінити визначення компонента компонентом у стилі класу, оскільки воно еквівалентне стилю об 'єкта звичайних опцій визначення компонента.

Визначаючи компонент у стилі класу, можна не тільки змінити синтаксис, а й використовувати деякі функції мови ECMAScript, такі як успадкування класу і декоратори. Компонент класу Vue також надає засіб підтримки mixins для успадкування mixin і функцію creceDecorator для простого створення власних декораторів.

```
<template>
  <div>
    <button v-on:click="decrement">-</button>
    {{ count }}
    <button v-on:click="increment">+</button>
  </div>
</template>
<script>
import Vue from 'vue'
import Component from 'vue-class-component'
// Define the component in class-style
```

```
@Component
export default class Counter extends Vue {
  // Class properties will be component data
  count = 0
  // Methods will be component methods
  increment() {
    this.count++
  }
  decrement() {
    this.count--
  }
}
</script>
```

1.6.3 vue-router

Vue Router - офіційна бібліотека маршрутизації для Vue.js. Вона глибоко інтегрується з Vue.js і дозволяє легко створювати SPA-додатки. Включає такі можливості:

- Вкладені маршрути/перегляди;
- Модульні налаштування маршрутизатора;
- Доступ до параметрів маршруту, query, wildcards;
- Анімація переходів подань на основі Vue.js;
- Зручний контроль навігації;
- Автоматичне проставляння активного CSS класу для посилань;
- Режими роботи HTML5 history або хеш, з авто-перемиканням в IE9;
- Налаштовувана поведінка прокрутки сторінки.

1.6.4 vuetify

VuetifyJS - це величезний пак готових компонентів для програми на VueJS. Кастомних компонентів настільки багато, що, спочатку, можна легко

заплутатися в них. Тут є і своя розмітка, і своя друкарня тощо. Загалом, повний набір для інтерфейсу програми.

Vuetify - це повна рамка інтерфейсу користувача, створена поверх Vue.js. Мета проекту - надати розробникам інструменти, необхідні для створення багатого і привабливого користувальницького досвіду. На відміну від інших рамок, Vuetify розроблений з самого початку, щоб бути простим у вивченні і винагороді освоїти сотні ретельно розроблених компонентів зі специфікації Material Design.

Vuetify використовує мобільний перший підхід до дизайну, що означає, що ваш додаток просто працює в готовому вигляді - будь то на телефоні, планшеті або настільному комп 'ютері.

Якщо ви досвідчений розробник і хочете порівняти Vuetify з іншими бібліотеками/фреймворками, ознайомтеся з нашими

З моменту свого початкового релізу в 2014 році Vue.js виріс до одного з найпопулярніших рамок JavaScript у світі. Однією з причин такої популярності є широке використання компонентів, які дозволяють розробникам створювати стислі модулі для використання і повторного використання у всьому їх застосуванні. Бібліотеки користувацького інтерфейсу представляють собою збірки цих додатків, які реалізують конкретні рекомендації щодо стилю і надають необхідні інструменти для створення розширених веб- програм.

Vuetify розробляється точно відповідно до специфікації Material Design, при цьому кожен компонент ретельно відпрацьовується, щоб бути модульним, чуйним і продуктивним. Налаштуйте програму за допомогою унікальних та динамічних компонувань та налаштуйте стилі компонентів за допомогою змінних SASS.

Vuetify має дуже активний цикл розробки і виправляється щотижня, реагуючи на проблеми спільноти і звіти з великою швидкістю, дозволяючи вам частіше отримувати доступ до виправлень помилок і поліпшень. Крім того, кожен великий реліз супроводжується 18-місячною довгостроковою підтримкою попередньої мінорної версії.

Висновок

Можна сказати, що Vuetify - це дуже роздутий інструмент, який ставати корисним після основного вивчення. Так що, будьте готові витратити 2-3 години на читання мануалів. В кінцевому результаті цей фронтенд-пак стає приємним фактором у розробці додатків. У ньому можна знайти готове рішення для інтерфейсу практично будь-якого завдання.

1.7 Vuex

Vuex - патерн управління станом + бібліотека для додатків на Vue.js. Він служить централізованим сховищем даних для всіх компонентів програми з правилами, що гарантують, що стан може бути змінено тільки передбачуваним чином. Vuex інтегрується з офіційним розширенням vue-devtools (opens new window), надаючи "з коробки" такі просунуті можливості, як "машину часу" для налагодження та експорт/імпорт зліпків стану даних.

```
new Vue({
 // стан
 data() {
  return {
   count: 0
  };
 },
 // перегляд
 template: `
```

```
    <div>{{ count }}</div>
  `,
  // дії
  methods: {
    increment() {
      this.count++;
    }
  }
});
```

Ця самостійна програма складається з таких частин:

1. Стан - "джерело істини", що управляє додатком;
2. Подання - відображення стану задане декларативно;
3. Дії - можливі шляхи зміни стану програми у відповідь на взаємодію користувача з поданням.

Однак простота швидко зникає, коли у нас з 'являється кілька компонентів, що ґрунтуються на одному і тому ж стані:

- Декілька відображень можуть залежати від однієї частини стану програми.
- Дії з різних уявлень можуть впливати на одні й ті самі частини стану програми.

Вирішуючи першу проблему, доведеться передавати одні й ті самі дані вхідними параметрами в глибоко вкладені компоненти. Це часто складно і втомливо, а для сусідніх компонентів таке і зовсім не спрацює. Вирішуючи другу проблему, можна прийти до таких рішень, як звернення за посиланнями до батьківських/дочірніх примірників або спроб змінювати і синхронізувати кілька копій стану через події. Обидва підходи крихки і швидко призводять до появи коду, який неможливо підтримувати.

Так чому б не винести весь загальний стан програми з компонентів і керувати ним в глобальному синглтоні? При цьому наше дерево компонентів стає одним великим "уявленням", і будь-який компонент може отримати доступ до стану програми або викликати дії для зміни стану, незалежно від того, де вони знаходяться в дереві!

Чітко визначаючи і поділяючи концепції, що виникають при управлінні станом, і вимагаючи дотримання певних правил, які підтримують незалежність між уявленнями і станами, ми краще структуруємо код і полегшуємо його підтримку.

Це основна ідея Vuex, натхненного Flux (opens new window), Redux (opens new window) і Архітектурою Elm (opens new window). На відміну від інших набоїв, Vuex реалізований у вигляді бібліотеки, спеціально призначеної для Vue.js, щоб використовувати його систему реактивності для ефективного оновлення.

1.7.1 Коли слід використовувати Vuex?

Vuex - патерн управління станом + бібліотека для додатків на Vue.js. Він служить централізованим сховищем даних для всіх компонентів програми з правилами, що гарантують, що стан може бути змінено тільки передбачуваним чином. Vuex інтегрується з офіційним розширенням vue-devtools (opens new window), надаючи "з коробки" такі просунуті можливості, як "машину часу" для налагодження та експорт/імпорт зліпків стану даних.

Vuex допомагає керувати спільно використовуваним станом ціною привнесення нових концепцій і допоміжного коду. Компроміс, коли короткочасна продуктивність страждає на благо довгострокової.

Якщо ще не доводилося створювати великі SPA і ви лише знайомитеся з Vuex, це може здатися багатослівним і складним. Все в порядку - прості програми можуть легко обходитися і без Vuex. Можливо, буде досить простого патерну глобальної шини подій (opens new window). Але якщо ви створюєте SPA середнього або великого розміру, то, швидше за все, вже

стикалися з ситуаціями, які змушували задуматися про те, як краще керувати станом поза компонентами Vue, а Vuex в такому випадку може стати цілком природним наступним кроком.

Є хороша цитата від Дена Абрамова, автора Redux:
«Flux-бібліотеки схожі на окуляри: ви будете точно знати, коли вони вам знадобляться».

1.7.2 Основні поняття

1.7.2.1 Єдине дерево стану

Vuex використовує єдине дерево стану - коли один об 'єкт містить весь глобальний стан програми і служить "єдиним джерелом істини". Це також означає, що в додатку буде тільки одне таке сховище. Єдине дерево стану дозволяє легко знайти потрібну його частину або робити знімки поточного стану програми з метою зневадження.

Єдине дерево стану не суперечить модульності - в наступних головах ми вивчимо, як можна розділити стан і мутації на під-модулі. Дані, які зберігаються у Vuex повинні слідувати тим же правилам, що і data в екземплярі Vue, тобто об 'єкт стану повинен бути простим.

1.7.2.2 Геттери

Іноді може знадобитися обчислювати похідний стан на основі стану сховища, наприклад, відфільтрувати список і потім підрахувати кількість елементів:

```
computed: {
  doneTodosCount () {
    return this.$store.state.todos.filter(todo => todo.done).length
  }
}
```

Якщо такі обчислення потрібні більш ніж в одному компоненті, доведеться або дублювати функцію, або виносити її в загальний метод, який потім імпортувати у всіх місцях - обидва підходи далекі від ідеалу.

Vuex дозволяє визначати "геттери" в сховищі. Ви можете вважати їх вирахованими властивостями сховища. Як і обчислювані властивості, результати геттера кешуються, на основі його залежностей і перераховуються тільки при зміні однієї з залежностей.

```
const store = new Vuex.Store({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos: state => {
      return state.todos.filter(todo => todo.done);
    }
  }
});
```

1.7.2.3 Мутації

Єдиним способом зміни стану сховища у Vuex є мутації. Мутації у Vuex дуже схожі на події: кожна мутація має рядковий тип і функцію-обробник. У цьому обробнику і відбуваються, власне, зміни стану, переданого до функції першим аргументом:

```
const store = new Vuex.Store({
  state: {
    count: 1
```

```
 },
 mutations: {
  increment(state) {
   // изменяем состояние
   state.count++;
  }
 }
});
```

Викликати функцію-обробник безпосередньо - не можна. Це більше схоже на обробку події: "Коли мутація типу increment ініційована, викликається цей обробник". Щоб ініціювати обробку мутації, необхідно викликати store.commit, вказавши її тип:

```
store.commit('increment');
```

1.7.2.4 Дії

Дії - схожі на мутації з кількома відмінностями: Замість того, щоб безпосередньо змінювати стан, дії ініціюють мутації; Дії можуть використовуватися для асинхронних операцій.

Зареєструємо просту дію:

```
const store = new Vuex.Store({
 state: {
  count: 0
 },
 mutations: {
  increment(state) {
   state.count++;
  }
 },
 actions: {
```

```
    increment(context) {
      context.commit('increment');
    }
  }
});
```

Обробники дій отримують об 'єкт контексту, що містить ті ж методи і властивості, що і сам екземпляр сховища, так що ви можете викликати context.commit для ініціювання мутації або звернутися до стану і геттерів через context.state і context.getters. Пізніше під час розгляду Модулів ми побачимо, однак, що цей контекст - не те ж саме, що екземпляр сховища.

На практиці для спрощення коду часто використовується деструктуризація аргументів (opens new window) з ES2015 (особливо при необхідності багаторазового виклику commit):

```
actions: {
  increment ({ commit }) {
    commit('increment')
  }
}
```

1.7.2.5 Додатки

Використовуючи єдиний дерево стану, всі стани програми містяться всередині одного великого об 'єкта. Однак, у міру зростання і масштабування програми, сховище може істотно роздутися.

Щоб допомогти в цій біді, Vuex дозволяє розділяти сховище на модулі. Кожен модуль може містити власний стан, мутації, дії, геттери і навіть вбудовані підмодулі - структура фрактальна:

```
const moduleA = {
  state: () => ({ ... }),
```

```
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}
const moduleB = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... }
}
```

1.8 Firebase

Firebase - це одне з BaaS-рішень (Backend as a Service), яке дає розробнику масу можливостей. Це і сервер, і база даних, і хостинг, і автентифікація в одній платформі. Так, Firebase Realtime Database надає розробникам API, який синхронізує дані програми між клієнтами і зберігає їх в хмарному сховищі. Програма підключається до бази даних через WebSocket, який відповідає за синхронізацію даних протягом усього сеансу.

Firebase - це платформа розробки мобільних додатків з величезним функціоналом. Починалася вона як стартап, а сьогодні її використовують при розробці кращих кроссплатформенних додатків. Головна перевага платформи в тому, що вона дозволяє розробнику не відволікатися на створення бекенду, тобто прихованої від користувача програмної частини проекту, наприклад, серверного коду. І це спрощує і прискорює створення мобільних додатків, дає можливість повністю зосередитися саме на UX/UI, тобто, на користувальницькому інтерфейсі та досвіді. Саме зв 'язка Firebase з фреймворком Flutter дозволяє програмістам компанії AVADA MEDIA створювати швидкі програми для Android і iOS, що дозволяють вирішувати різні завдання.

Firebase може забезпечити підтримку вашої програми, включаючи зберігання даних, автентифікацію користувачів, статичний хостинг і багато

іншого. Зосередьтеся на створенні незвичайного власного досвіду. Ми подбаємо про інше. Створюйте кроссплатформенні власні мобільні та веб-програми за допомогою наших Android, iOS і JavaScript SDK. Ви також можете підключити Firebase до існуючого бекенду за допомогою наших серверних бібліотек або нашого REST API.

Особливості Firebase

- База даних у реальному часі - Firebase підтримує дані JSON, і всі підключені до неї користувачі отримують оперативні оновлення після кожної зміни.
- Автентифікація - ми можемо використовувати анонімну, парольну або іншу соціальну автентифікацію.
- Хостинг. Програми можуть бути розгорнуті через захищене з 'єднання з серверами Firebase.

Переваги Firebase

- Це просто і зручно для користувача. Немає потреби у складній конфігурації.
- Дані в режимі реального часу, що означає, що кожна зміна буде автоматично оновлювати з 'єднаних клієнтів.
- Firebase пропонує просту панель керування.
- Є ряд корисних послуг на вибір.

Обмеження Firebase

Безкоштовний план Firebase обмежений 50 підключеннями і 100 МБ сховища.

РОЗДІЛ 2

Чому варто використовувати typescript у сучасних веб-програмах

2.1 Що таке JavaScript? Коротка історія

JavaScript (також відомий як ECMAScript) почав своє життя як проста скриптова мова для браузерів. У той час, коли він був винайдений, його передбачалося використовувати для коротких фрагментів коду, вкладених у веб-сторінку - написання більше декількох десятків рядків коду було б дещо незвичним. Завдяки цьому ранні веб-браузери досить повільно виконували такий код. Однак з часом JS став все більш популярним, і веб-розробники почали використовувати його для створення інтерактивних можливостей.

Розробники веб-браузера відреагували на це зростання використання JS, оптимізувавши свої виконавчі механізми (динамічна компіляція) і розширивши те, що з ним можна зробити (додавання API), що в свою чергу змусило веб-розробників використовувати його ще більше. На сучасних веб-сайтах браузер часто запускає програми, які охоплюють сотні тисяч рядків коду. Це тривалий і поступовий ріст "веб", що починається як проста мережа статичних сторінок, і еволюціонує в платформу для багатих додатків усіх видів.

Більш того, JS став досить популярним для використання поза контексту браузерів, наприклад, для реалізації серверів JS за допомогою node.js. "Біжучий куди завгодно" характер JS робить його привабливим вибором для кроссплатформенної розробки. Зараз існує багато розробників, які використовують тільки JavaScript для програмування всього їх стека!

Підбиваючи підсумок, ми маємо мову, яка була розроблена для швидкого використання, а потім зросла до повноцінного інструменту для написання додатків з мільйонами рядків. Кожна мова має свої причуди - дивацтва і сюрпризи, і скромний початок JavaScript змушує її мати багато таких. Деякі приклади:

Оператор рівності JavaScript (= =) створює аргументи, що призводить до несподіваної поведінки:

```
if ("" == 0) {
//Це так! Але чому?
}
if (1 < x < 3) {
//True для * будь-якого * значення x!
}
```

JavaScript також дозволяє отримати доступ до відсутніх властивостей:

```
const obj = { width: 10, height: 15 };
// Чому це NaN?
const area = obj.width * obj.heigth;
```

У більшості мов програмування виникає помилка при виникненні таких помилок, деякі роблять це під час компіляції - до запуску будь-якого коду. При написанні невеликих програм такі причуди дратують, але керовані; при написанні додатків із сотнями або тисячами рядків коду ці постійні сюрпризи є серйозною проблемою.

2.2 TypeScript: перевірка статичного типу

Раніше ми говорили, що деякі мови взагалі не дозволять цим баггі-програмам працювати. Виявлення помилок у коді без його виконання називається статичною перевіркою. Визначення того, що є помилкою, а що не засновано на видах оброблюваних значень, називається перевіркою статичного типу.

TypeScript перевіряє програму на наявність помилок перед виконанням, і робить це на основі типів значень, це засіб перевірки статичного типу. Наприклад, у останньому прикладі вище виявлена помилка через тип об 'єкта.

Ось виявлена помилка TypeScript:

```
const obj = { width: 10, height: 15 };
const area = obj.width * obj.heigth;
```

Властивість "heigth" "не існує в типі" "{ширина: number; висота: число;}".
Типізований суперсет JavaScript

2.2.1 Синтаксис

TypeScript є мовою, яка є супернабором синтаксису JavaScript: JS, тому є допустимим TS. Синтаксис означає спосіб написання тексту для формування програми. Наприклад, цей код має синтаксичну помилку, оскільки в ньому відсутня a):

```
let a = (4
')'  Очікувався
```

TypeScript не вважає код JavaScript помилкою через свій синтаксис. Це означає, що ви можете взяти будь-який працюючий код JavaScript і помістити його в файл TypeScript, не турбуючись про те, як саме він написаний.

2.3 Типи

Однак TypeScript є типізованим суперсетом, що означає додавання правил використання різних типів значень. Більш рання помилка про obj.heigth не була синтаксичною помилкою: це помилка використання якогось значення (типу) неправильно.

В якості іншого прикладу можна навести код JavaScript, який можна запустити в оглядачі і записати в журнал значення:

```
console.log(4 / []);
```

Ця синтаксично легальна програма реєструє нескінченність. TypeScript, однак, вважає поділ числа на масив безглуздою операцією і видає помилку:

Права частина арифметичної операції повинна бути типу "any" ", number" "," bigint "" або типу перерахування.

Можливо, ви дійсно мали намір розділити число на масив, можливо, просто щоб побачити, що відбувається, але в більшості випадків це помилка програмування. Засіб перевірки типу TypeScript призначений для виправлення програм при виявленні максимально можливої кількості поширених помилок. (Пізніше ми дізнаємося про параметри, які можна використовувати для налаштування суворої перевірки коду за допомогою TypeScript).

Під час переміщення коду з файлу JavaScript у файл TypeScript можуть з 'явитися помилки при введенні залежно від способу запису коду. Це можуть бути законні проблеми з кодом або занадто консервативний TypeScript. У цьому підручнику буде показано, як додати різні синтаксиси TypeScript для усунення таких помилок.

Поведінка під час виконання

TypeScript також є мовою програмування, яка зберігає поведінку JavaScript під час виконання. Наприклад, ділення на нуль у JavaScript створює Infinity замість створення винятку під час виконання. У принципі TypeScript ніколи не змінює поведінку коду JavaScript під час виконання.

Це означає, що при переміщенні коду з JavaScript в TypeScript він буде виконуватися аналогічним чином, навіть якщо TypeScript вважає, що код має помилки типу.

Збереження тієї ж поведінки під час виконання, що і JavaScript, є основною обіцянкою TypeScript, оскільки це означає, що ви можете легко переходити між двома мовами, не турбуючись про тонкі відмінності, які можуть змусити вашу програму перестати працювати.

2.3.1 Стерті типи

Грубо кажучи, як тільки компілятор TypeScript завершує перевірку коду, він стирає типи для створення результуючого "скомпільованого" коду. Це

означає, що після компіляції коду отриманий простий код JS не містить інформації про тип.

Це також означає, що TypeScript ніколи не змінює поведінку програми залежно від її типів. Суть у тому, що, хоча при компіляції можуть спостерігатися помилки типу, сама система типів не впливає на роботу програми при її запуску.

Нарешті, TypeScript не надає додаткових бібліотек середовища виконання. У програмах використовуватиметься та ж стандартна бібліотека (або зовнішні бібліотеки), що і в програмах JavaScript, тому для вивчення не потрібна додаткова інфраструктура TypeScript.

TypeScript має незвичайне відношення до JavaScript. TypeScript пропонує всі функції JavaScript і додатковий шар поверх них: система типів TypeScript.

Наприклад, JavaScript надає примітиви мови, такі як послідовність і число, але він не перевіряє, що вони присвоєні послідовно. TypeScript робить. Це означає, що існуючий робочий код JavaScript також є кодом TypeScript. Основна перевага TypeScript полягає в тому, що він може виділяти несподівану поведінку в коді, знижуючи ймовірність помилок.

2.3.2 Типи виводів

TypeScript знає мову JavaScript і в багатьох випадках створює типи. Наприклад, коли ви створюєте змінну та її призначення, TypeScript використовуватиме значення як власний тип.

```
let helloWorld = "Hello World";
let helloWorld: string
```

Розуміючи, як працює JavaScript, TypeScript може побудувати систему типів, яка приймає код JavaScript, але має типи. Це пропонує систему типів без додавання додаткових символів, щоб зробити типи явними в коді. Саме тому TypeScript знає, що helloWorld є послідовністю в наведеному вище прикладі.

Можливо, ви написали JavaScript у Visual Studio Code і отримали автозавершення редактора. Visual Studio Code використовує TypeScript під ковпаком, щоб спростити роботу з JavaScript.

2.4 Визначення типів

У JavaScript можна використовувати різноманітні візерунки дизайну. Однак деякі візерунки проектування ускладнюють автоматичне визначення типів (наприклад, візерунки, які використовують динамічне програмування). Щоб охопити ці випадки, TypeScript підтримує розширення мови JavaScript, яке пропонує вам місця, щоб повідомити TypeScript, які типи повинні бути.

Наприклад, щоб створити об 'єкт з видимим типом, що включає назву: послідовність та ідентифікатор: число, можна записати:

```
const user = {
  name: "Hayes",
  id: 0,
};

interface User {
  name: string;
  id: number;
}
```

Потім можна оголосити, що об 'єкт JavaScript відповідає формі нового інтерфейсу, використовуючи синтаксис, наприклад: TypeName після оголошення змінної:

```
const user: User = {
  name: "Hayes",
  id: 0,
};
```

Якщо ви надаєте об 'єкт, який не відповідає наданому інтерфейсу, TypeScript попереджає:

```
interface User {
  name: string;
  id: number;
}
const user: User = {
  username: "Hayes",
Type '{ username: string; id: number; }' is not assignable to type 'User'.
  Object literal may only specify known properties, and 'username' does not exist in type 'User'.
  id: 0,
};
```

Оскільки JavaScript підтримує класи та об 'єктно-орієнтоване програмування, так само як і TypeScript. Оголошення інтерфейсу можна використовувати з класами:

```
interface User {
  name: string;
  id: number;
}
class UserAccount {
  name: string;
  id: number;
  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}
const user: User = new UserAccount("Murphy", 1);
```

Для анотації параметрів та повернення значень до функцій можна використовувати інтерфейси:

```
function getAdminUser(): User {
  //...
}
function deleteUser(user: User) {
  // ...
}
```

JavaScript вже має невеликий набір примітивних типів: boolean, bigint, null, number, послідовність, symbol і undefined, які можна використовувати в інтерфейсі. TypeScript розширює цей список ще кількома, такими як будь-який (дозволити що-небудь), невідомий (переконайтеся, що хтось, який використовує цей тип, оголошує, що це тип), ніколи (неможливо, щоб цей тип міг статися) і void (функція, яка повертає невизначене або не має поверненого значення).

Ви побачите, що є два синтаксиси для побудови типів: Інтерфейси і Типи. Ви повинні віддати перевагу інтерфейсу. Використовуйте тип, коли вам потрібні певні функції.

2.5 Створення типів

За допомогою TypeScript можна створювати складні типи, комбінуючи прості типи. Є два популярних способи зробити це: з "Союзами" і з "Дженериками".

Союзи

За допомогою об 'єднання можна оголосити, що тип може бути одним з багатьох типів. Наприклад, можна описати логічний тип як true або false:

```
type MyBool = true | false;
```

Примітка: Якщо ви наведете курсор на MyBool вище, ви побачите, що він класифікується як boolean. Це властивість системи структурного типу. Детальніше про це нижче.

Популярним прикладом використання для типів об 'єднання є опис набору послідовностей або літералів чисел, які можуть мати значення:

```
type WindowStates = "open" | "closed" | "minimized";
type LockStates = "locked" | "unlocked";
type OddNumbersUnderTen = 1 | 3 | 5 | 7 | 9;
```

Профспілки також забезпечують спосіб обробки різних типів. Наприклад, у вас може бути функція, яка приймає масив або послідовність:

```
function getLength(obj: string | string[]) {
  return obj.length;
}
```

Щоб дізнатися тип змінної, скористайтеся типом: рис 2.0

Type	Predicate
string	`typeof s === "string"`
number	`typeof n === "number"`
boolean	`typeof b === "boolean"`
undefined	`typeof undefined === "undefined"`
function	`typeof f === "function"`
array	`Array.isArray(a)`

Рис 2.9 типи змінних

2.6 Переваги TypeScript

Коли ми говоримо про переваги TypeScript, на думку зазвичай приходить наступний список:

- TypeScript підтримує статичну типізацію
- TypeScript робить код простішим для читання і розуміння
- TypeScript допомагає уникнути безлічі болючих багів, які зазвичай здійснюють розробники, завдяки перевірці типів в коді
- TypeScript заохочує розробників слідувати кращим ОВП практикам
- Як наслідок перерахованого вище - TypeScript економить час розробників

РОЗДІЛ 3
PWA і SPA програми та їхні особливості

Тенденція створення односторінкових додатків (SPA) і прогресивних веб-додатків (PWA) поступово набирає все великих обертів у веб-індустрії. Обґрунтовано це масою переваг, серед яких відносно проста розробка, висока швидкість і зручність для користувачів, а також нові можливості розвитку бізнесу.

3.1. Що таке SPA і PWA

SPA (Single Page Application) - односторінковий веб-додаток, що завантажується на одній HTML-сторінці і не вимагає перезавантаження за рахунок динамічного оновлення за допомогою AJAX. Робота SPA виконується всередині браузера, а відмальовка коду відбувається на боці користувача.

Популярні приклади динамічних сайтів - Gmail, Google Maps, Facebook, Meduza. Під час переходу на них спочатку вам видається основний контент, а для виконання різних дій (прокрутка, перехід на інші сторінки) замість перезавантаження сторінок завантажуються лише окремі елементи.

До переваг односторінкових додатків належать такі аспекти:

- висока швидкість порівняно з традиційними сайтами;
- добре налагоджена робота як на десктопних, так і на мобільних пристроях;
- гнучкість і чуйність за рахунок відсутності перезавантаження і повторного рендерінгу;
- оптимізована і спрощена розробка.

Недоліки:

- збільшене навантаження на браузер;
- відсутність JS може стати проблемою для багатьох функцій;

- необхідність забезпечити хороший захист даних - через міжсайтовий скриптинг (XSS) зловмисники можуть отримати доступ до коду і додавати скрипти на боці клієнта.

PWA (Progressive Web Apps) - це сайт, побудований на веб-технологіях за допомогою JavaScript, HTML і CSS і взаємодіє з відвідувачем як додаток. Він може додаватися на головний екран мобільних пристроїв і відправляти push-повідомлення. За рахунок доступу до апаратних засобів пристрою PWA працює без з 'єднання з інтернетом. Наочним прикладом PWA-сайту служить Google Docs, який відомий як онлайн-офіс, але користувач може взаємодіяти з ним і в автономному режимі.

Плюси PWA-сайтів:

- простота в розробці: створення PWA - це не створення окремого сайту, а мінімальна зміна існуючого;
- миттєва установка;
- кроссплатформенність;
- мінімальні вимоги до ресурсів пристроїв;
- швидке завантаження онлайн і в автономному режимі завдяки кешуванню сторінок;
- велика швидкість порівняно з нативними додатками.

Мінуси

- не всіма браузерами підтримуються всі функції;
- складність у пошуку розробників: PWA - нова і поки ще не дуже поширена тенденція, і тому далеко не всі розробники зможуть з нею впоратися.

3.1.1 Відмінності між сайтами типу SPA і PWA

Багато хто вважає, що веб-програми (PWA) - це і є односторінкові програми (SPA), що зовсім неправильно. SPA може бути PWA, але PWA зовсім не обов 'язково повинен бути SPA.

3.1.2 Принцип роботи динамічного відображення

При роботі динамічних сайтів сервер розпізнає пошукових роботів, а запити від них передаються засобу відображення. Запити від клієнта обробляються так само, як і для традиційних сайтів. При потребі динамічний відображення може повертати версію контенту, яку робот може обробити, наприклад, статичну HTML-сторінку.

Принцип роботи PWA-сайтів описується наступним чином:

Під час першого відвідування користувачем PWA-сайту встановлюється скрипт Service Worker, який додає оболонку програми в кеш. Після завантаження оболонки програма запитує вміст для заповнення перегляду, а потім запитується контент. Коли всі запити завершені, service worker переходить в режим очікування і знаходиться там до тих пір, поки мережевий запит не ініціює нову подію.

3.4 Особливості кешування

Для односторінкових можна налаштувати різні типи кешування:

DOM-сховище, в якому зберігаються вже сформовані фрагменти HTML-коду. DOM-сховище може бути двох видів: локальне і сеансове. У сеансовому сеансі зберігаються дані сеансу і після завершення вилучаються, а в локальному дані кешуються назавжди.

HTTP-кешування. Цей тип кешування застосовується, щоб зберігати відповіді сервера на стороні клієнта, тим самим уникаючи повторних запитів. Кешування на сервері. Завдяки цьому типу знижується навантаження на сервер, оскільки всі динамічні запити підтягуються з власного кешу.

Кешування PWA-сайтів реалізується шляхом впровадження однієї з наступних стратегій:

- Кешування активів - дозволяє використовувати одну оболонку програми, коли вона знаходиться в автономному режимі, або кешувати дані вручну при взаємодії з клієнтом.
- Кешування з резервом - користувач отримує відповідь, що знаходиться в кеші, онлайн-запити не виконуються.
- Відкат у кеші - при такій стратегії клієнт завжди виходить актуальну версію сайту, якщо використовує додаток онлайн. В офлайн-режимі користувачі отримують кешовану версію.

Щоб не виникати проблем з кешуванням, незалежно від реалізованої стратегії, для PWA використовується файл manifest.json, що містить JSON-форматовані властивості (ім 'я, іконки, кольори, URL головної сторінки).

3.5. SPA-сайти

Основна трудність в оптимізації SPA-сайтів полягає в індексації. Наприклад, боти пошукової системи Яндекс не можуть обробити JavaScript: щоб проіндексувати сторінки, їм необхідно наявність її повної HTML-копії.

Щоб повідомити роботам про HTML-копію сторінки, необхідно використовувати? _ escaped _ fragment _ = в URL разом з метатегом meta name = "fragment" content = "!", які вказуються в коді динамічної сторінки. Наприклад, для індексу сторінки http://example.com/page1/боту потрібно надати її копію: http://example.com/page1/?_escaped_fragment_=.

Існує також варіант з використанням "#!", якщо адреса AJAX формується за допомогою "#". Наприклад, URL https://example.com/# url потрібно змінити на https://example.com/#!url.

Якщо ви не хочете розгортатися з параметрами URL-адрес, спочатку вибирайте кадр, який підтримує серверний рендеринг.

Google, у свою чергу, використовує клієнтський рендеринг за допомогою браузера Chrome, і йому не потрібні HTML-копії сторінок для індексації. Коли Google-бот заходить на сторінку, спочатку він виконує аналіз, а потім визначає час для рендерингу. Те саме відбувається, якщо скрипти на сторінці занадто складні для обробки. Тому для спрощення індексації сайту пошуковими роботами краще створювати бібліотеки JavaScript - поліфіли, які використовуються для написання кодів і працюють у всіх браузерах.

Для просування SPA-сайтів використовуйте ті ж методи, що і для традиційних. Щоб сайт добре сприймався ботами, необхідно:

- не використовувати вікна iframe;
- використовувати статичні URL;
- займатися оптимізацією скриптів для збільшення швидкості їх завантаження.

3.6 особливості СПА

На цей час SPA- і PWA-сайти досить популярні, і цьому є пояснення:

- вони прості в розробці;
- забезпечують швидку швидкість роботи;
- позитивно впливають на користувацький досвід.

Однак поки, на жаль, не всі браузери і пошукові системи можуть правильно обробити дані, тому при впровадженні цих технологій подбайте про наступні моменти:

- правильно налаштовуйте URL-адреси;
- створюйте HTML-копії сторінок;
- перевіряйте на коректність коди відповіді сервера;
- після змін на сторінках перевіряйте працездатність скриптів (наприклад, за допомогою Netpeak Spider).

Інші ж аспекти оптимізації SPA- і PWA-сайтів нічим не відрізняються від оптимізації традиційних сайтів.

РОЗДІЛ 4

Опис веб додатка

4.1 Опис “Customers”

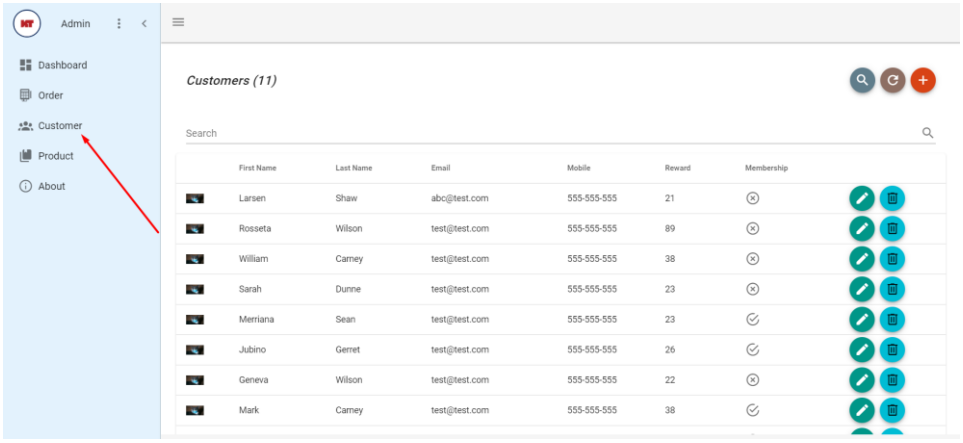

Рис 4.10 Опис Customers

У вкладці клієнта можна

1. Зробити пошук
2. Фільтрувати за алфовітом
3. Редагувати
4. Видаляти
5. Створювати

Пошук

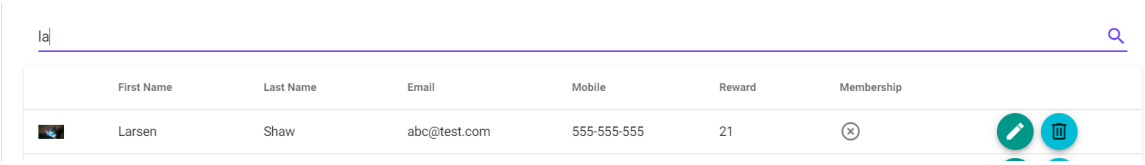

Рис 4.11 Пошук

Фільтрувати за алфовітом

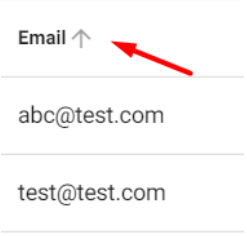

Рис 4.12 приклад фільтрації

Редагування

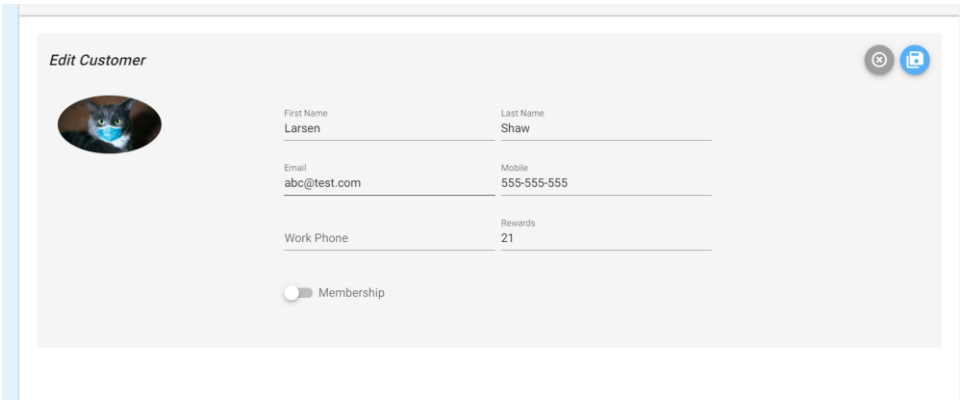

Рис 4.13 Редагування Customers

Створення клієнта відбувається в цій вкладці, можна додати всю потрібну інформацію про клієнта і так само відзначити галочку membership.

Видаляти

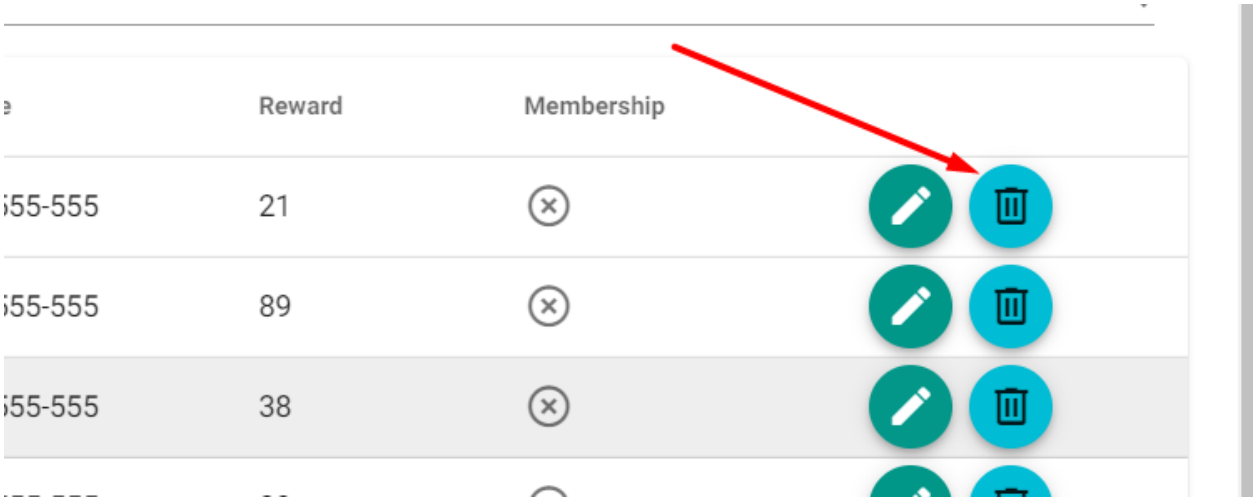

Рис 4.14 Видалення Customers

Створювати

Рис 4.15 Створення Customers

4.2 Опис “Product”

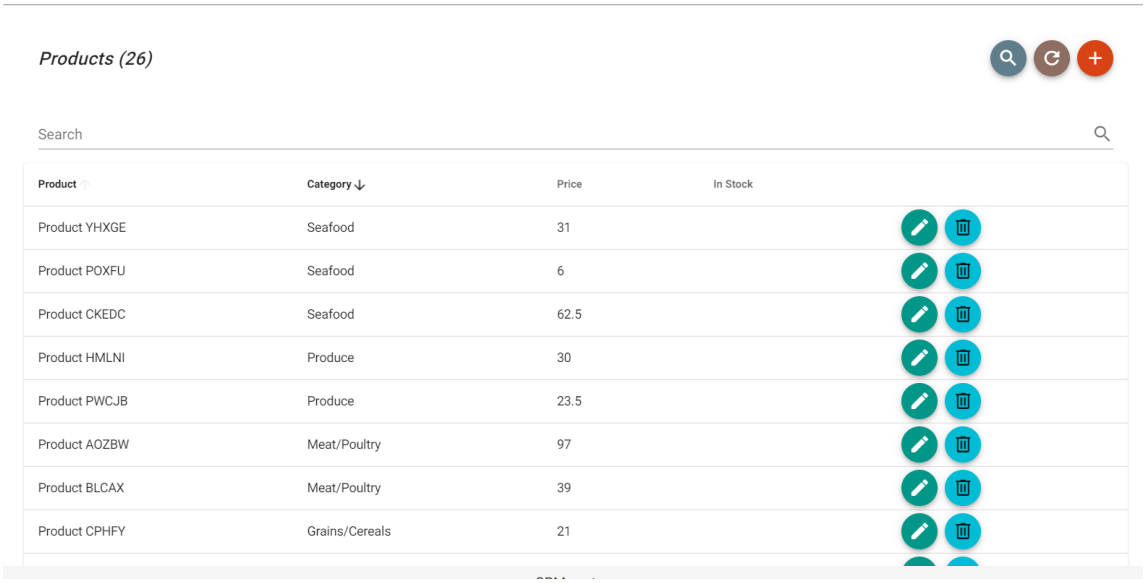

Рис 4.16 Тип об'єкта продукт

1. Зробити пошук
2. Фільтрувати за алфовітом
3. Редагувати
4. Видаляти
5. Створювати

Пошук

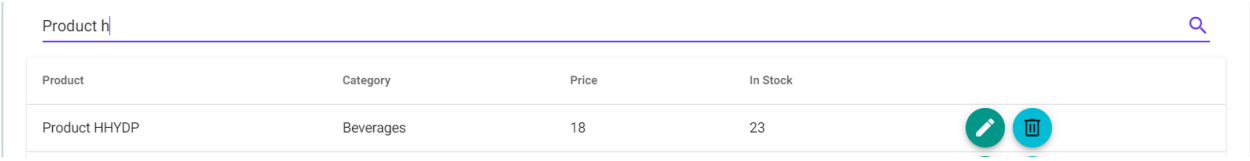

Рис 4.17 Пошук

Фильтрувати

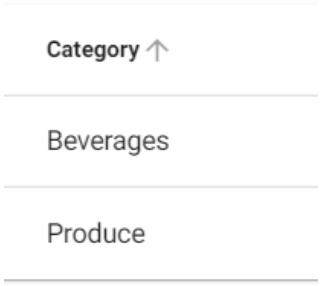

Рис 4.18 приклад фільтрації

Редагувати

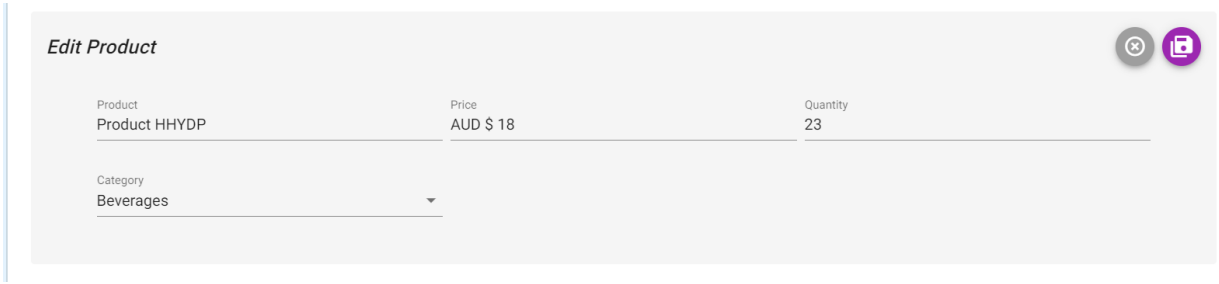

Рис 4.19 редагування product

Так само можна продукт прив'язати до категорії.

Створювати

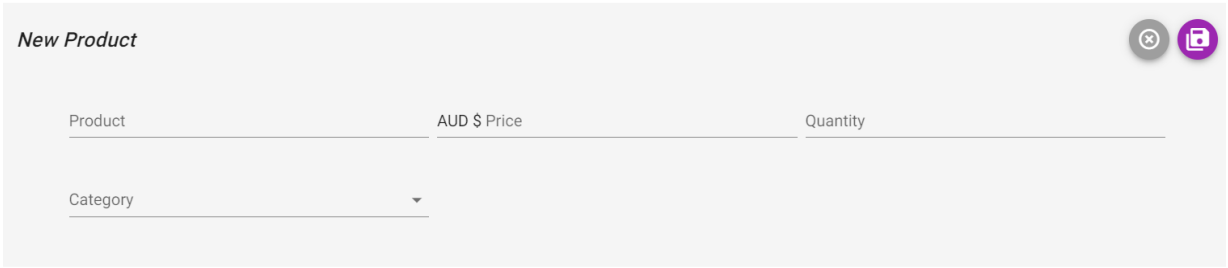

Рис 4.20 Створювання product

4.3 Опис “Orders”

1. Зробити пошук
2. Фільтрувати за алфавітом
3. Редагувати
4. Видаляти
5. Створювати

Пошук

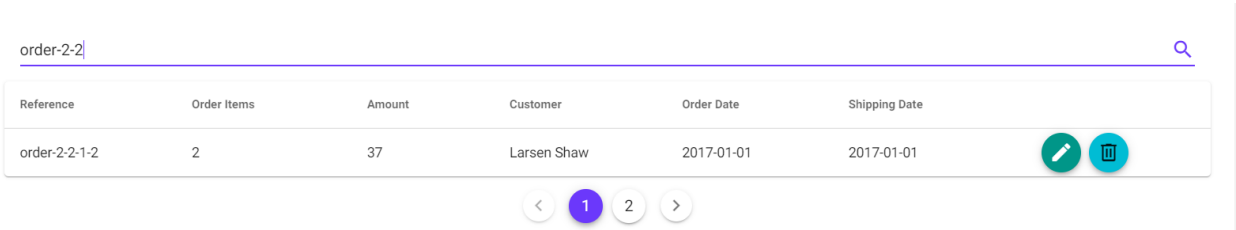

Рис 4.21 Пошук

Фільтрувати

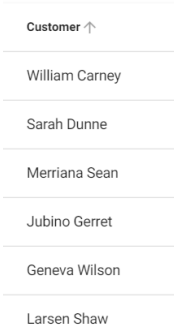

Рис 4.22 приклад фільтрації

Редагувати

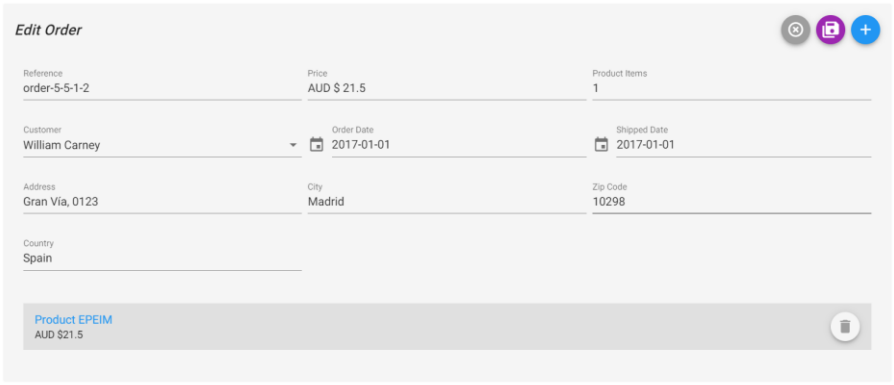

Рис 4.23 редагування замовлення

Видаляти

Рис 4.24 Видалити продукт

Створювати

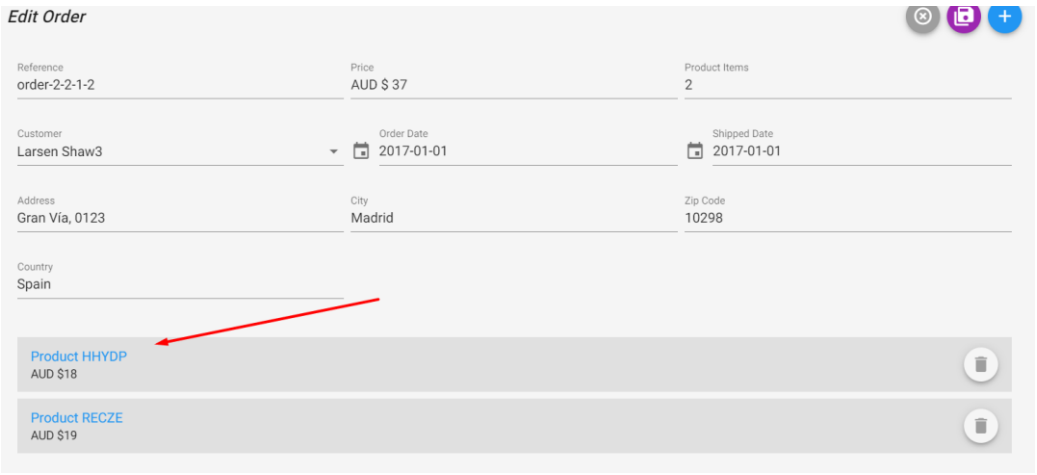

Рис 4.25 Створення продукт

Так само до нього прив'язані товари

4.4 About

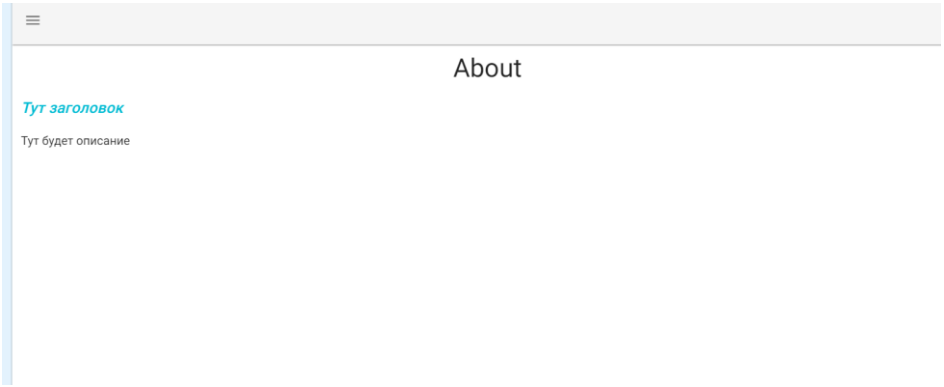

Рис 4.26 Опис у SRM

Тут виводиться опис компанії.

4.5 Головна

Рис 4.27 Відображення графіків

На головній виводяться графіки з инофрмацию яка передається по апі. При кліці на меню можна змінити пароль або вийти з облікового запису.

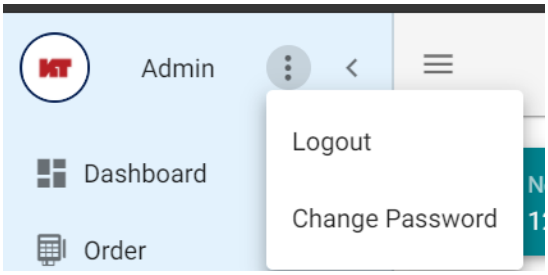

Рис 4.28 Вихід з облікового запису

Висновок

Основна суть роботи була така, це розгляд SPA та typescript та їх переваг.

- Промальовує Контент не повністю, а компонентно це те що потрапляє на екран користувача
- Відносно легкий поріг входження, для розробки і тестування досить встановленої node.js та npm
- Значно високе завантаження контенту яка економить ресурси
- SPA швидкі. Перехід між модулями в додатку відбувається швидше: потрібні ресурси вже завантажені, потрібно просто підставити дані, які запросив користувач. Часто при цьому сервер повертати не великоваговий HTML, а легкий JSON або XML.
- Ще використання JSON спрощує розробку програми для різних платформ. Якщо для веб-версії розробити звичайний сайт, який приймає від сервера HTML, то для мобільного додатка доведеться писати доопрацювання, так як там HTML не підійде. JSON робить відповідь сервера універсальним.
- SPA гнучкі. Раз користувач весь час працює з однією сторінкою, простіше робити цікаві переходи і анімацію елементів. Можна працювати зі станом кнопок, вкладок і перемикачів. Таким чином, інтерфейс SPA може бути схожий скоріше на повноцінний додаток, а не на простий сайт.
- SPA працюють всюди. Все, що потрібно для SPA - підтримка JavaScript. Такі сайти добре працюють і на робочому столі, і в інтернеті, можуть частково замінити повноцінні мобільні додатки.

Приклади динамічних додатків: Gmail, Google Maps, Facebok, GitHub, Meduza.

TypeScript - це надмножество JavaScript, тобто, будь-який код на JS є правильним з точки зору TypeScript. Однак, TypeScript володіє деякими додатковими можливостями, які не входять до JavaScript. Серед них - строга типізація (тобто, вказівка типу змінної при її оголошенні, що дозволяє зробити поведінку коду більш передбачуваним і спростити налагодження), механізми об'єктноорієнтованого програмування і багато іншого.

TypeScript поліпшить ваші навички як розробника JavaScript завдяки:

- Більшій впевненості перехоплення помилок до того, як вони потраплять у виробництво;
- Більш простому рефакторингу коду;
- Економії часу на написання тестів;
- Кращого досвіду кодування;
- Менше помилок у виробництві;
- Менше модульних тестів.

Часто за допомогою TypeScript код вашого партнера по команді говорить сам за себе. Їм не потрібно пояснювати його вам, тому що типи додають контекст до коду. Ці функції дозволяють довіряти команді більше. Ви працюєте на більш високому рівні, тому що проводите менше часу, турбуючись про дурних помилках. Це працює так само і для вашого коду. TypeScript змушує вас писати явний код. Побічним ефектом є миттєве підвищення якості коду.

Та це здатність TypeScript надавати можливість писати «майбутній» JavaScript. Зазвичай для цього нам потрібно кілька плагінів Babel. TypeScript, водночас час, надає цю ж функцію, але за рахунок однієї залежності. Команда TypeScript відмінно справляється зі специфікацією ECMAScript, додаючи мовні можливості Stage 3 і вище. Це означає, що ви можете використовувати нові доповнення до JavaScript, не заважаючи безлічі залежностей або конфігурації. Це дозволить вам випередити своїх колег по JavaScript. Обидві ці функції в поєднанні підвищать вашу ефективність в якості розробника JavaScript.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Axios та порівняння з featch (Електрон. Ресурс) / Спосіб доступу: URL: https://habr.com/ru/company/ruvds/blog/477286/
2. Опис роботи з Chart.js (Електрон. Ресурс) / Спосіб доступу: URL: https://ruseller.com/lessons.php?rub=32&id=2852
3. Core.js (Електрон. Ресурс) / Спосіб доступу: URL: https://habr.com/ru/post/216997/
4. Services worker (Електрон. Ресурс) / Спосіб доступу: URL: https://developer.mozilla.org/ru/docs/Web/API/Service_Worker_API/Using_Service_Workers
5. Vue.js (Електрон. Ресурс) / Спосіб доступу: URL: https://habr.com/ru/post/329452/ https://vuejs.org/ https://habr.com/ru/post/329452/ https://ela.kpi.ua/bitstream/123456789/40895/1/Riazantsev_bakalavr.pdf https://www.c-sharpcorner.com/article/an-overview-of-vue-js-frontend-framework/ https://www.gitmemory.com/issue/vuejs/vue-next/629/734378820
6. Vuetify (Електрон. Ресурс) / Спосіб доступу: URL: https://v2.vuetifyjs.com/ru/introduction/why-vuetify/
7. Vuex (Електрон. Ресурс) / Спосіб доступу: URL: https://vuex.vuejs.org/ru/ https://habr.com/ru/post/421551/ https://medium.com/@KucherDev/vuex-%D1%8D%D1%82%D0%BE-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE-6807d9ad330d https://essuir.sumdu.edu.ua/bitstream-download/123456789/79683/1/Kovalenko_bac_rob.pdf;jsessionid=A56FD3F8A4D9E49ADCA6A36590AACB7B
8. Fierbase (Електрон. Ресурс) / Спосіб доступу: URL: https://coderlessons.com/tutorials/veb-razrabotka/izuchite-firebase/firebase-kratkoe-rukovodstvo https://essuir.sumdu.edu.ua/bitstream-

download/123456789/79683/1/Kovalenko_bac_rob.pdf;jsessionid=A56FD3F8A4D9E49ADCA6A36590AACB7B

9. Typescript (Електрон. Ресурс) / Спосіб доступу: URL: https://senior.ua/articles/raznica-mezhdu-javascript-i-typescript https://webformyself.com/chto-takoe-typescript-staticheskaya-tipizaciya-dlya-interneta/ https://habr.com/ru/post/482702/https://habr.com/ru/company/ruvds/blog/344502/ https://dspace.nau.edu.ua/bitstream/NAU/45192/1/%D0%A4%D0%9A%D0%9A%D0%9F%D0%86_2020_122_%D0%A0%D0%BE%D0%BC%D0%B0%D0%BD%D0%B5%D0%BD%D0%BA%D0%BE_%D0%86%D0%9E.pdf
10. Порівняння PWA та SPA (Електрон. Ресурс) / Спосіб доступу: URL: https://webpromoexperts.net/blog/razbiraemsya-s-spa-i-pwa/
11. CRM system (Електрон. Ресурс) / Спосіб доступу: URL: https://salesap.ru/cr m_sistemy_chto_eto/ https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B7%D0%B0%D0%B8%D0%BC%D0%BE%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8_%D1%81_%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8 https://habr.com/ru/company/trinion/blog/249633/
12. SPA (Електрон. Ресурс) / Спосіб доступу: URL: https://vc.ru/seo/108149-odnostranichnye-spa-i-mnogostranichnye-pwa-veb-prilozheniya https://thecode.media/spa/