МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Сучасні каркаси web-додатків»

Лабораторна робота № 3

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

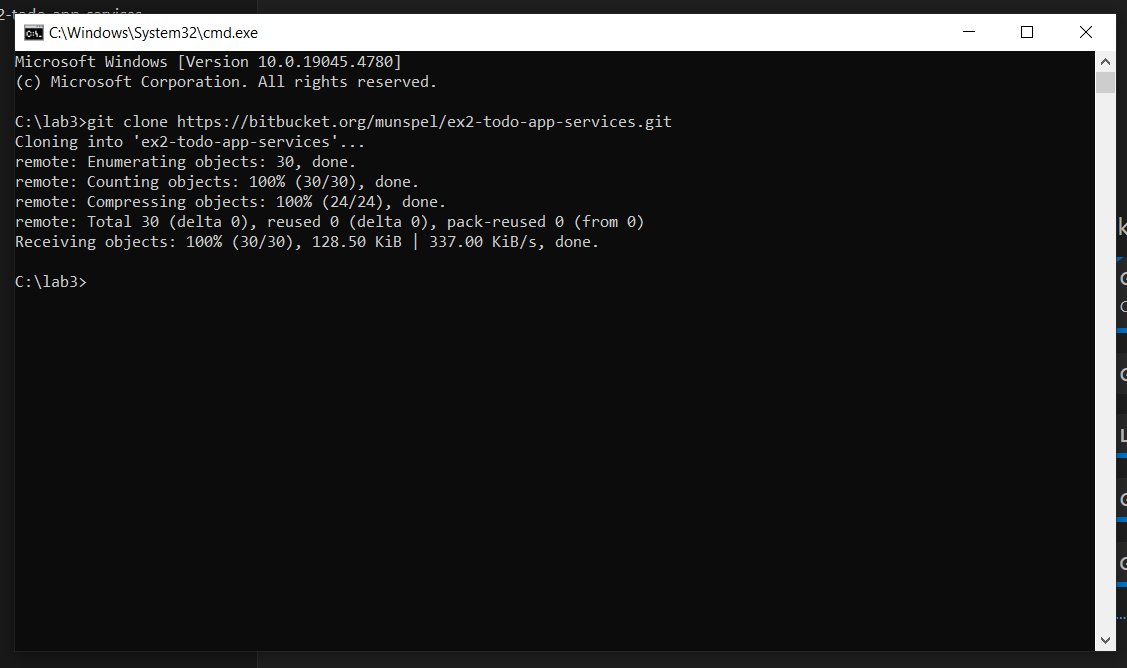
Перевірив: Лебідь Максим   
 Володимирович

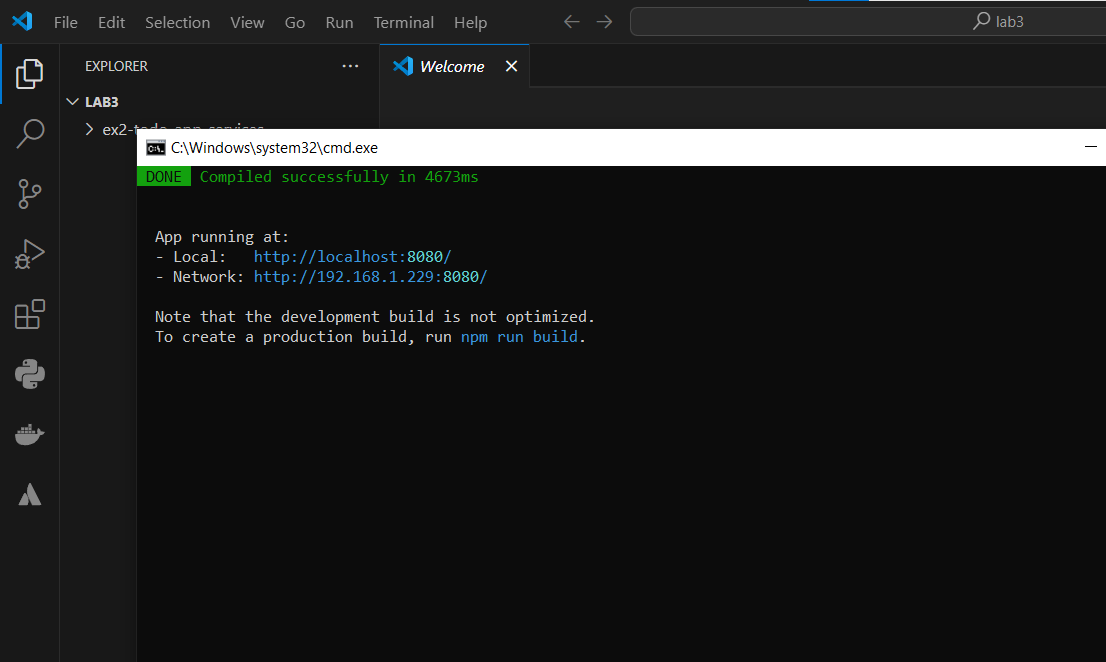
Запоріжжя  
2024

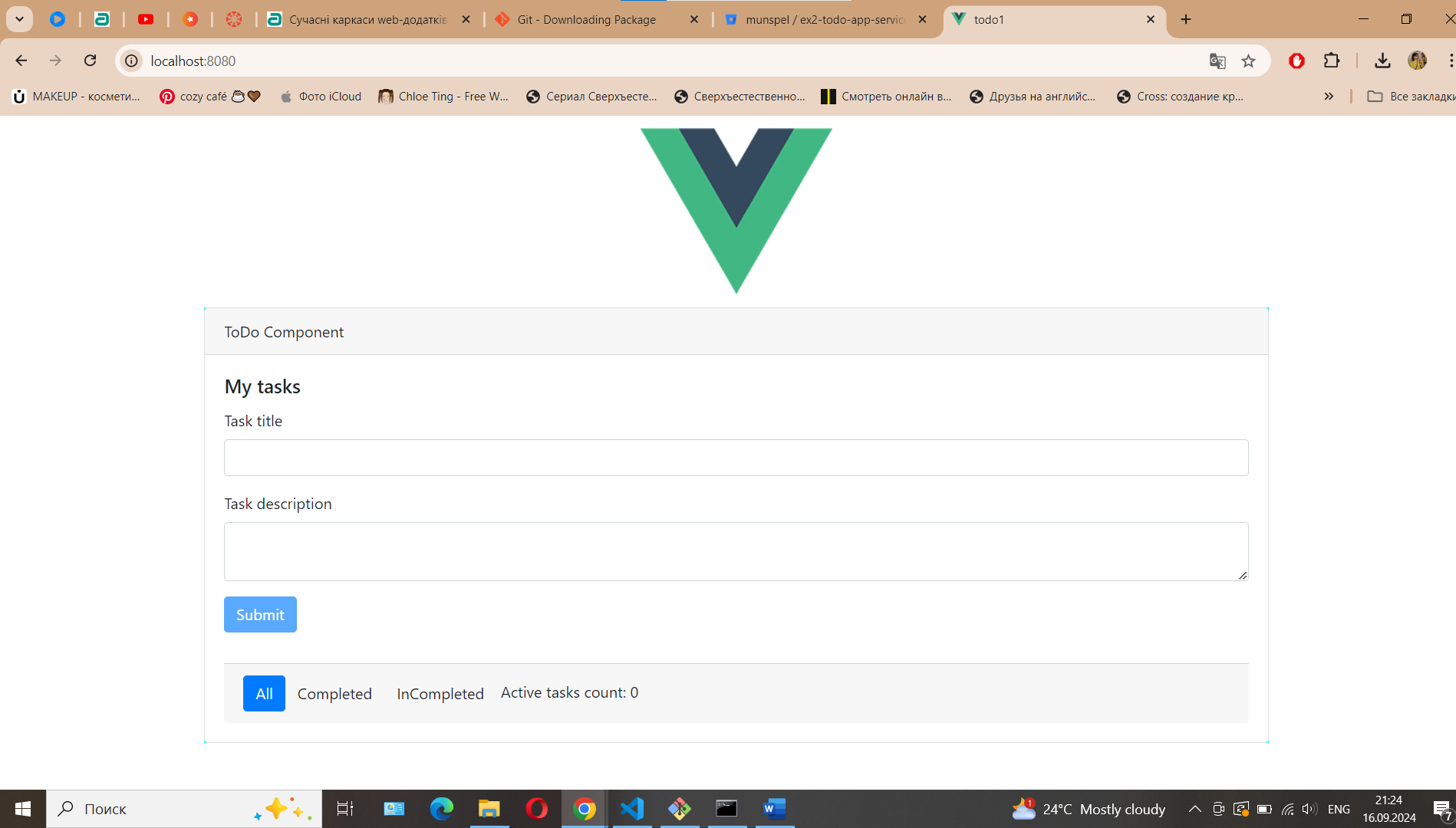
**Тема: Створення репозиторієв та сервісів**

**Завдання**

1. Виконати клонуванння та запуск проекту: <https://bitbucket.org/munspel/ex2-todo-app-services/src/master/>







2. Проаналізувати код з попереднього пункту та додаткові матеріали виконати (використовуючи TypeScript):

Розробити типи сутностей відповідно до потреб попередньої роботи.

Створіть компонент Button за наступними функціональними вимогами:

**Потреби до попередньої роботи, частина I:**

Приймати наступні пропси:

label: Текст кнопки.

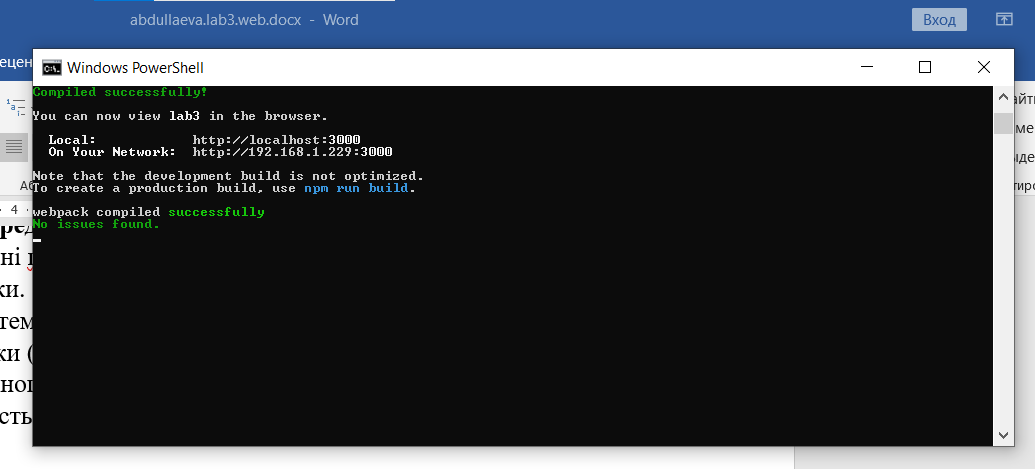
color: Кольорова тема кнопки (наприклад, червона, синя).

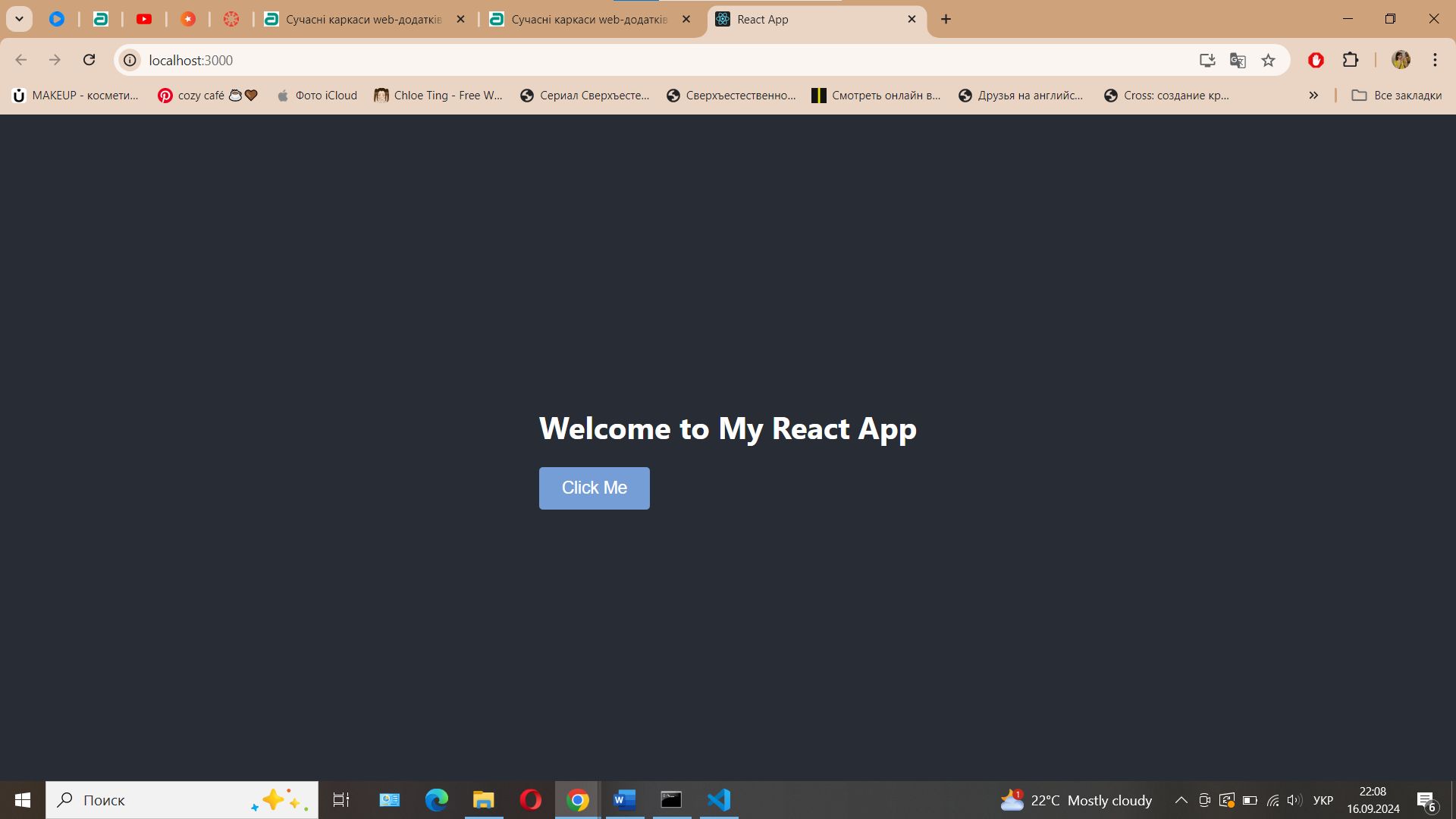
size: Розмір кнопки (маленька, середня, велика).

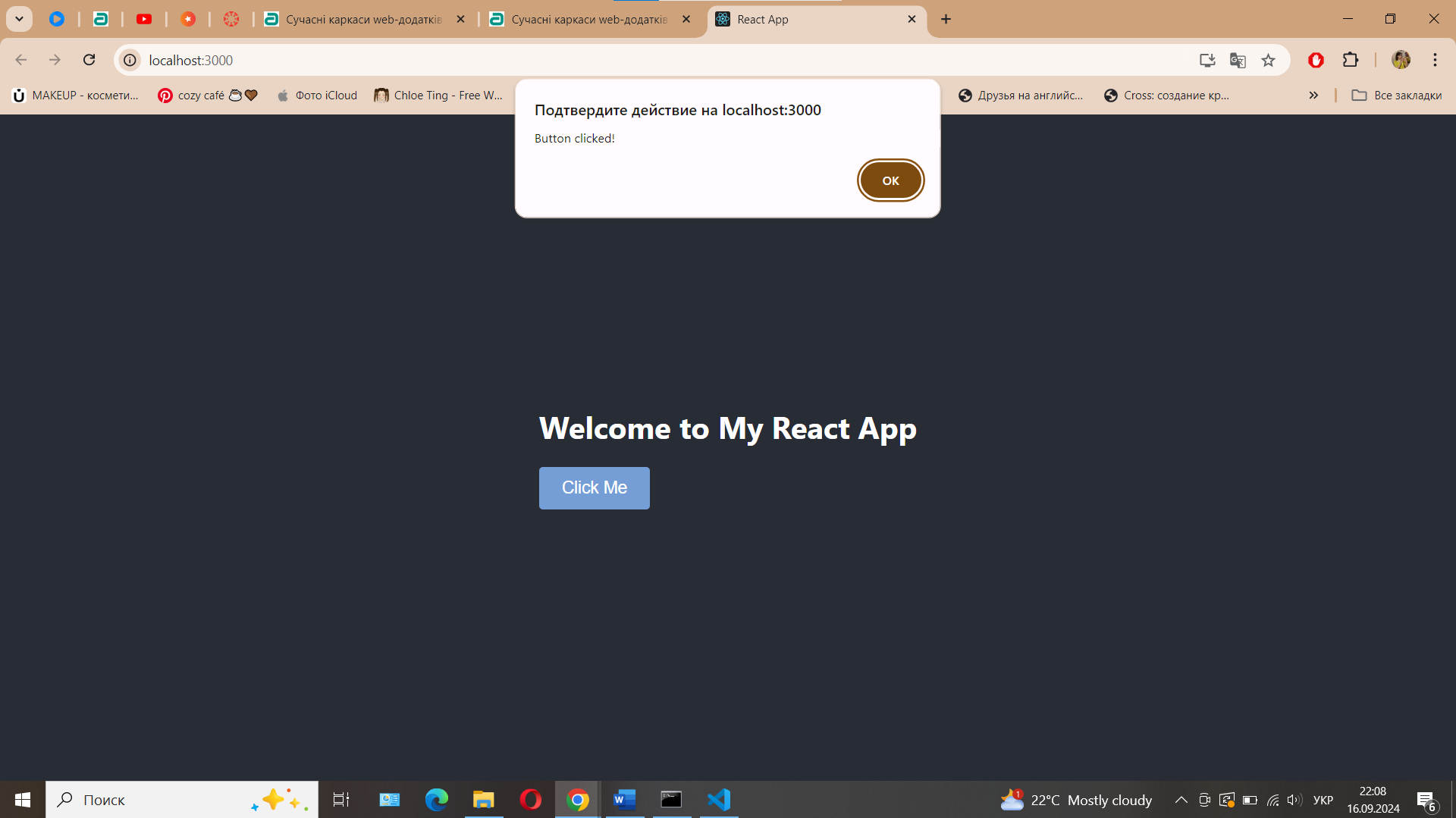
icon: Іконка для кнопки (необов’язкова).

Додайте можливість обробки події click на кнопці.

Виконаємо всі потрібні підготовки, та запустимо проект:







**Потреби до попередньої роботи, частина II:**

- Загальні вимоги:

* Приймає масив елементів для випадаючого списку через пропс.
* Має поле пошуку для фільтрації варіантів, що відображається при отриманні фокусу вводу.
* Використайте Tailwind для стилізації випадаючого списку та поля пошуку.
* Забезпечте можливість вибору елементу та передачі вибраного значення у батьківський компонент через подію.

- Пошук по значенням (Searchable):

* Користувач має можливість вводити текст для фільтрації елементів списку.
* Список варіантів має оновлюватися відповідно до введеного тексту в режимі реального часу.
* Якщо жоден варіант не відповідає введеному тексту, відображається повідомлення "Нічого не знайдено".

- Вибір значення (Selection):

* Користувач може вибрати одне або кілька значень зі списку (в залежності від налаштувань компонента).
* Після вибору список закривається, а вибране значення відображається в полі введення.

- Підтримка клавіатури:

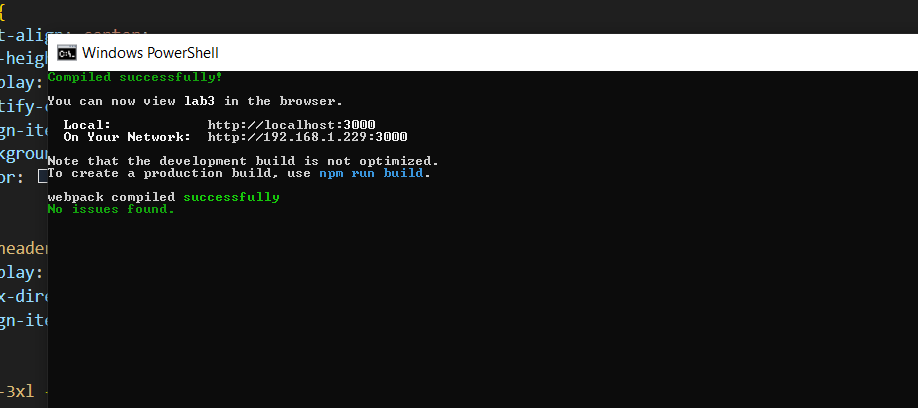
* Компонент підтримує навігацію за допомогою клавіш стрілок та Enter для вибору елементів.

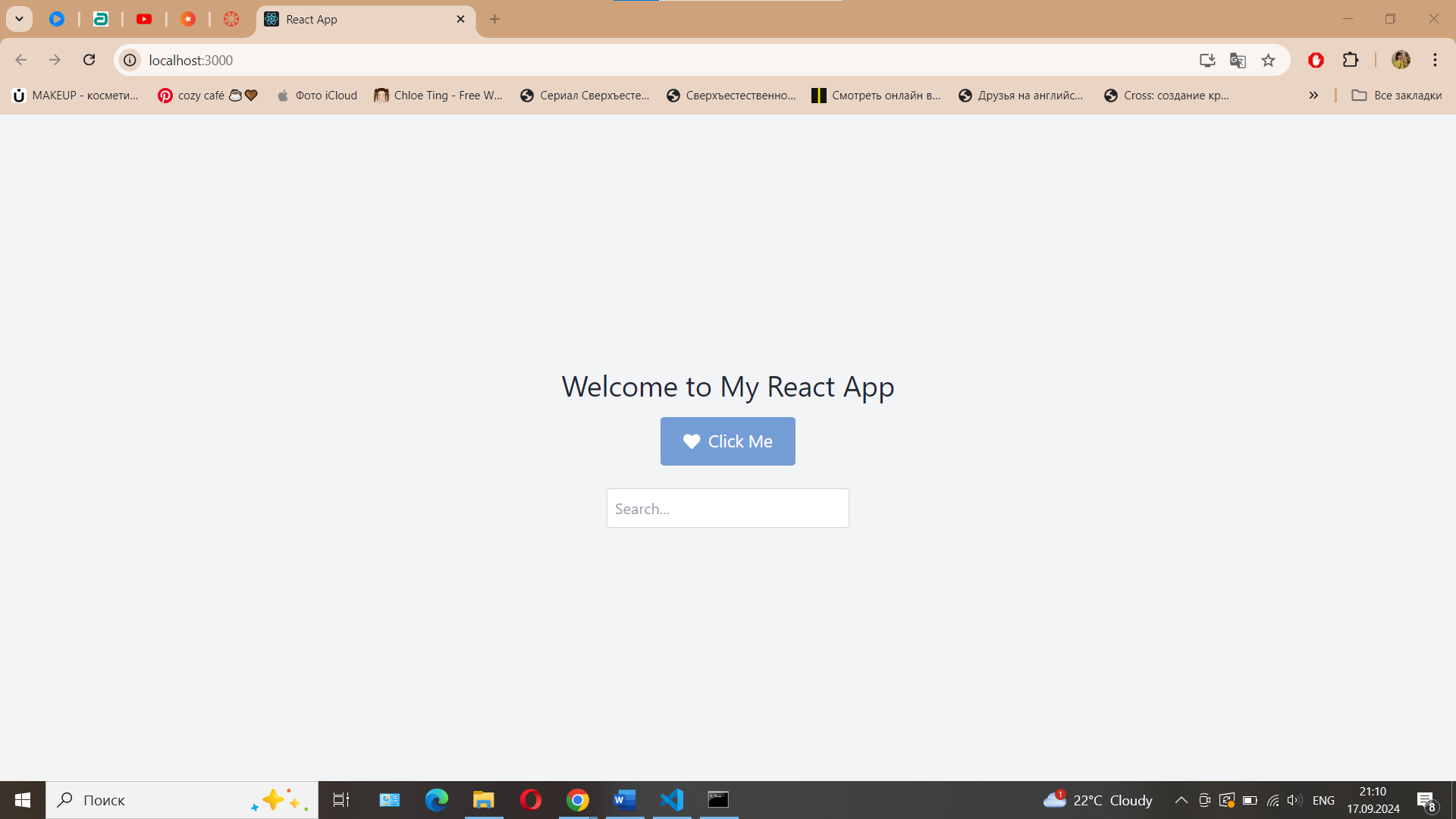
- Скидання вибору (Clear Selection):

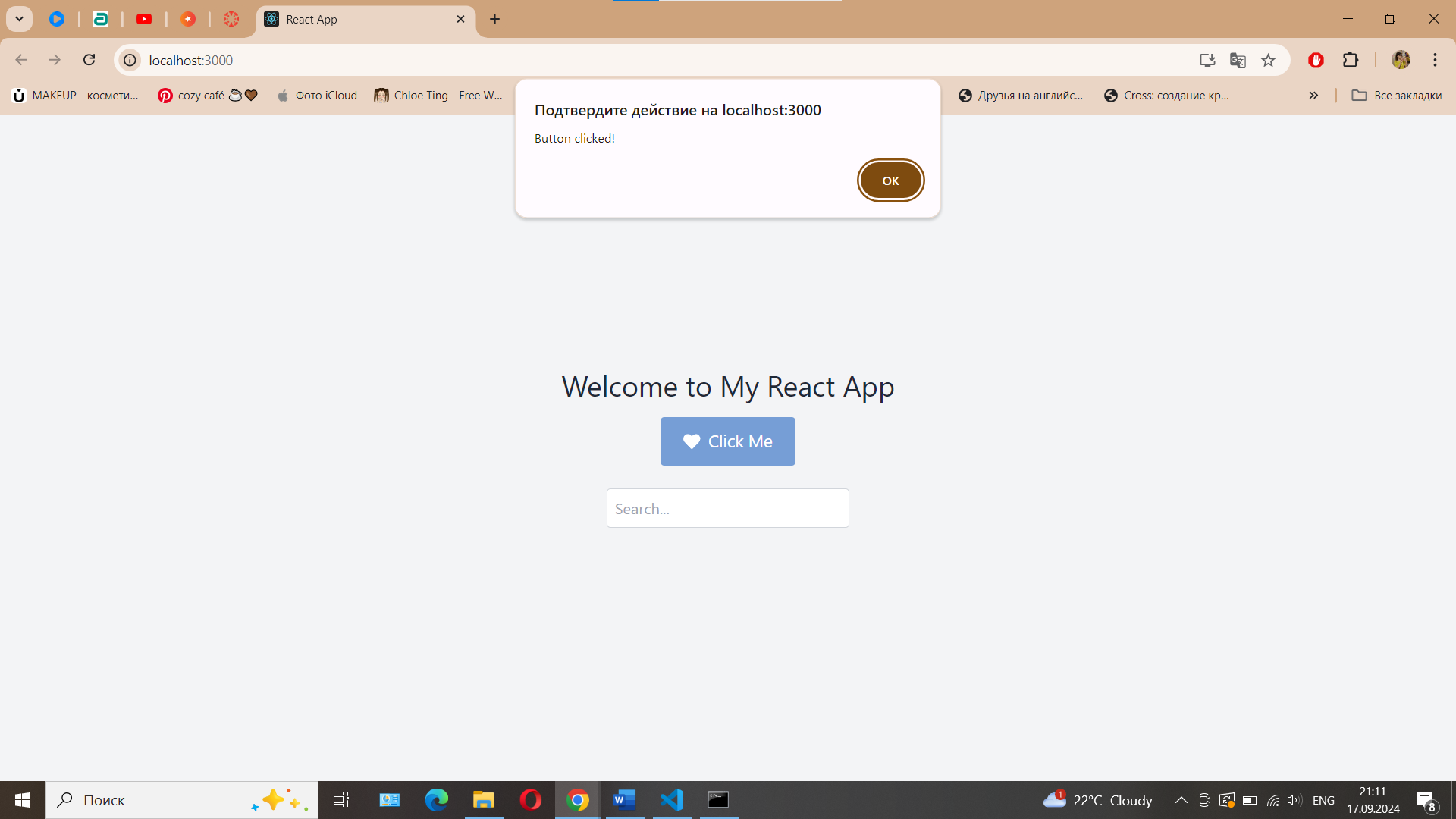
* Додати можливість очищати вибране значення через кнопку "очистити".

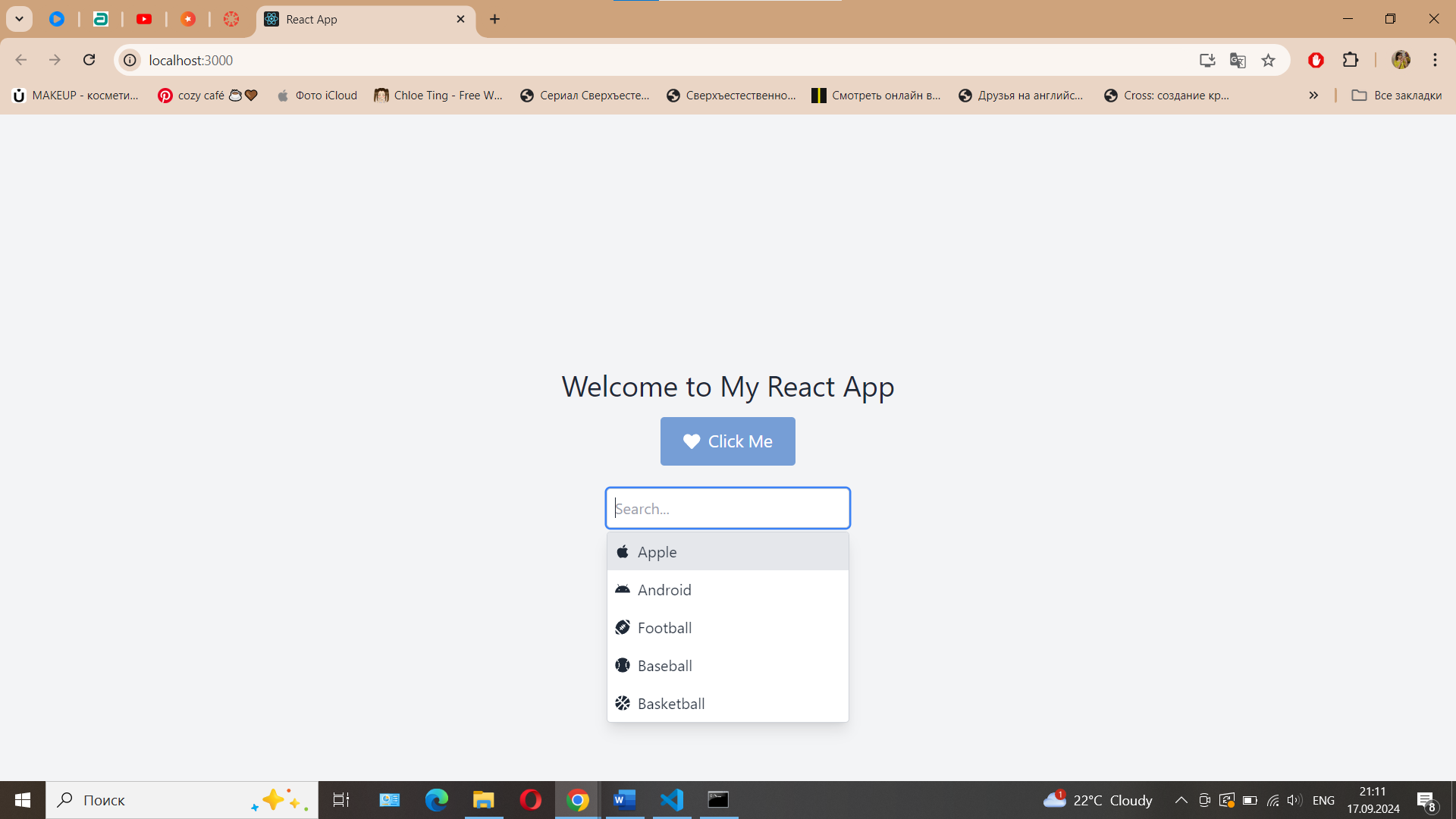
- Кастомізація відображення варіантів:

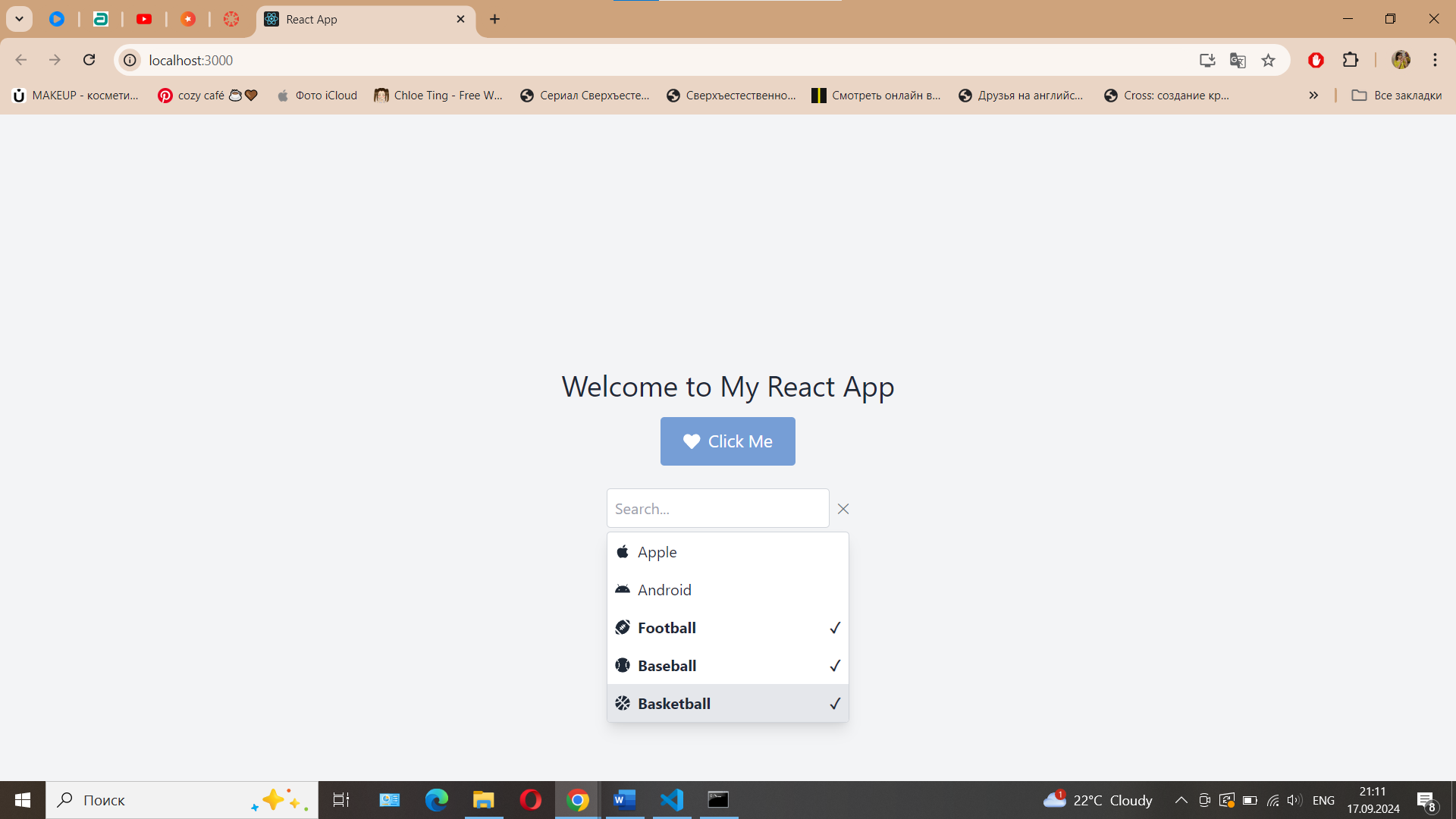
* Користувач може кастомізувати відображення елементів списку (наприклад, додавати іконки).

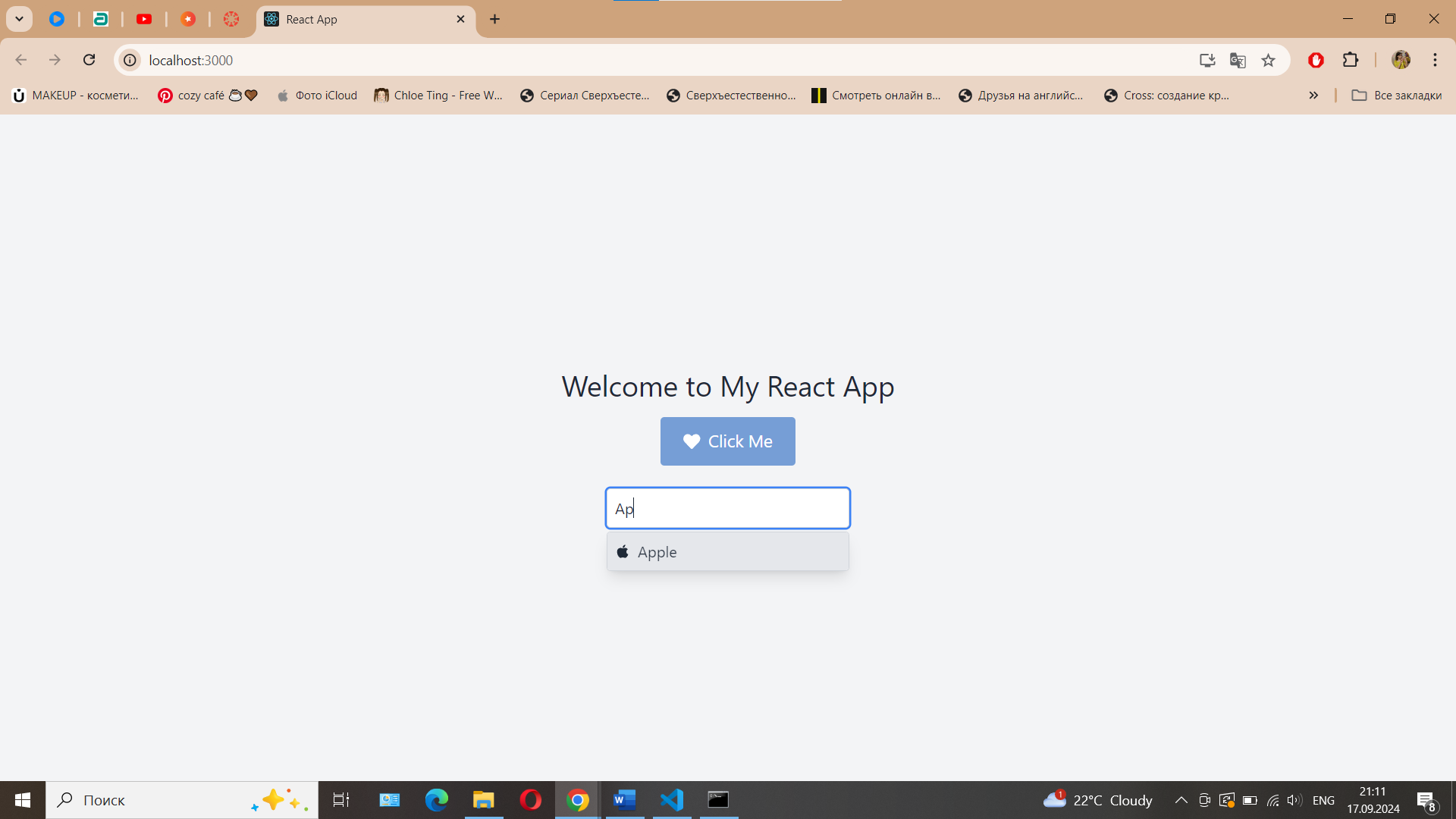




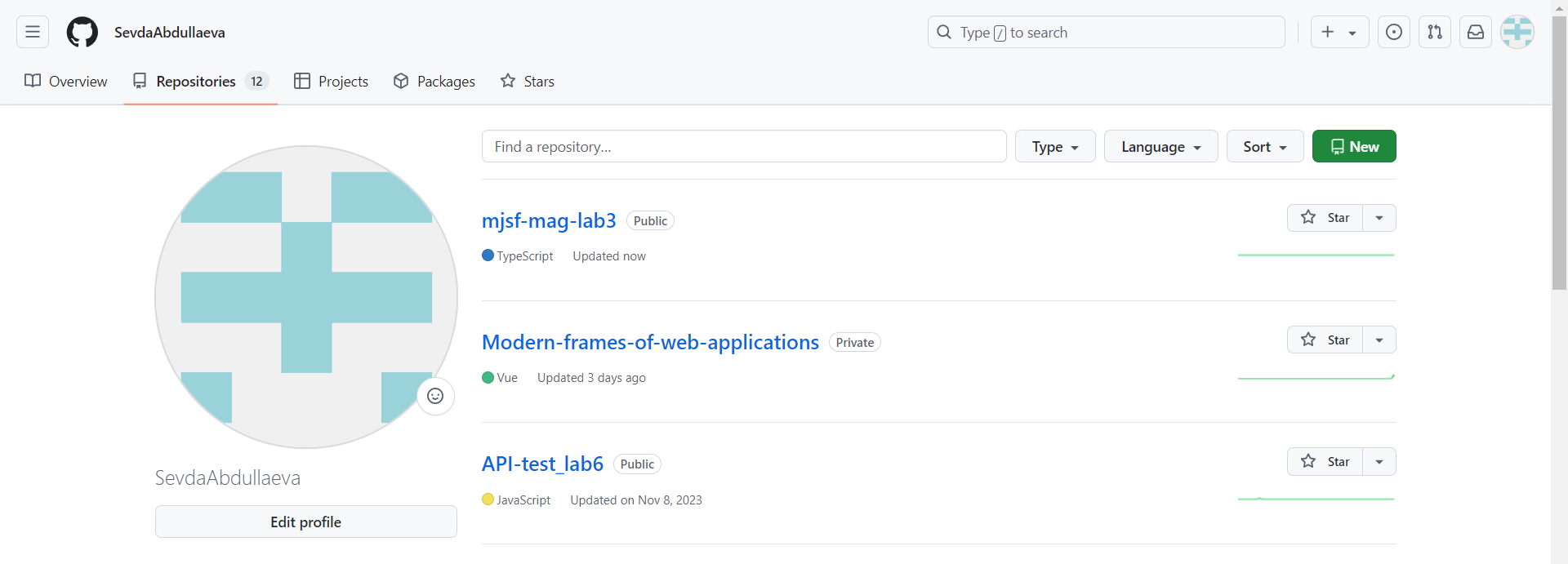








3. Розмістити проект у системі контролю версій (github.com, bitbucket.org) з іменем ропозиторію mjsf-mag-lab3.



4. Відправити посилання на репозиторій у якості звіту з виконаної роботи.

<https://github.com/SevdaAbdullaeva/mjsf-mag-lab3.git>

**Контрольні питання**

**1. Що таке props у Vue.js і для чого вони використовуються?**

props у Vue.js — це властивості, які передаються з батьківського компонента в дочірній. Вони використовуються для передачі даних і налаштувань від одного компонента до іншого. Зокрема, props дозволяють дочірнім компонентам отримувати дані та інформацію з батьківського компонента, щоб відобразити їх або використовувати в своїй логіці.

Ось основні моменти, що стосуються props:

* Передача даних: props дозволяють передавати дані з батьківського компонента в дочірній. Це може бути будь-який тип даних, наприклад, рядки, числа, масиви чи об'єкти.
* Налаштування компонентів: Вони часто використовуються для налаштування поведінки або вигляду дочірніх компонентів.
* Односторонній зв'язок: Дані передані через props є односторонніми. Це означає, що батьківський компонент передає дані, але дочірній не може змінювати ці дані безпосередньо. Для зміни даних зазвичай використовуються події або інші механізми.
* Валідація: props можуть бути валідаційними, що дозволяє задати правила перевірки для переданих даних, наприклад, типи даних або обов'язковість.
* Дефолтні значення: Можна задати дефолтні значення для props, які використовуватимуться, якщо дані не були передані.

**2. Які типи даних можна використовувати для пропсів у Vue? Як визначити тип пропсу?**

У Vue.js для props можна використовувати різні типи даних, і ви можете визначити тип prop для забезпечення коректності переданих даних. Основні типи даних, які можна використовувати для props, включають:

* String: Для рядкових значень.
* Number: Для числових значень.
* Boolean: Для логічних значень (правда/неправда).
* Array: Для масивів.
* Object: Для об'єктів.
* Function: Для функцій.
* Symbol: Для символів.

Щоб визначити тип prop, ви можете використовувати об'єкт у вашому компоненті. В об'єкті ви можете вказати тип prop, а також додаткові опції, такі як чи є prop обов'язковим і яке дефолтне значення має бути використане. Ось як це робиться:

<script>

export default {

props: {

// Рядковий тип

text: {

type: String,

required: true // Зазначає, що prop є обов'язковим

},

// Числовий тип з дефолтним значенням

numberValue: {

type: Number,

default: 0 // Дефолтне значення, якщо не передано

},

// Логічний тип

isActive: {

type: Boolean,

default: false

},

// Масив

items: {

type: Array,

default: () => [] // Дефолтне значення у вигляді масиву

},

// Об'єкт

userInfo: {

type: Object,

default: () => ({}) // Дефолтне значення у вигляді порожнього об'єкта

},

// Функція

onAction: {

type: Function,

default: () => () => {} // Дефолтне значення у вигляді пустої функції

}

}

};

</script>

Як це працює:

* type: Вказує тип даних, який prop повинен приймати.
* required: (Необов'язкове) Зазначає, чи є prop обов'язковим. Якщо так, а prop не передано, Vue видасть попередження.
* default: (Необов'язкове) Вказує дефолтне значення для prop, якщо значення не було передано батьківським компонентом.

Завдяки цьому ви можете забезпечити, щоб компоненти отримували саме ті дані, які від них очікуються, і зменшити ймовірність помилок.

**3. Як задати значення за замовчуванням для пропса у Vue?**

Щоб задати значення за замовчуванням для prop у Vue.js, ви використовуєте властивість default у об'єкті, що описує prop. Значення, вказане у default, буде використано, якщо значення для цього prop не передано з батьківського компонента.

Ось як це робиться:

* Просте значення (рядок, число, логічний тип):

<script>

export default {

props: {

// Рядковий тип з дефолтним значенням

text: {

type: String,

default: 'Default text'

},

// Числовий тип з дефолтним значенням

count: {

type: Number,

default: 10

},

// Логічний тип з дефолтним значенням

isActive: {

type: Boolean,

default: true

}

}

};

</script>

* Масив або об'єкт:

Для масивів і об'єктів потрібно використовувати функцію, яка повертає дефолтне значення. Це робиться для того, щоб уникнути проблем з спільним використанням об'єктів між різними екземплярами компонентів.

<script>

export default {

props: {

// Масив з дефолтним значенням

items: {

type: Array,

default: () => [] // Повертає новий порожній масив

},

// Об'єкт з дефолтним значенням

userInfo: {

type: Object,

default: () => ({}) // Повертає новий порожній об'єкт

}

}

};

</script>

* Функція:

<script>

export default {

props: {

// Функція з дефолтним значенням

onAction: {

type: Function,

default: () => () => {} // Повертає нову пусту функцію

}

}

};

</script>

У цьому прикладі для prop типів Array, Object, і Function важливо використовувати функцію, яка повертає дефолтне значення, щоб уникнути проблем із спільним використанням значень між різними екземплярами компонентів.

**4. Що таке emits у Vue.js і для чого вони використовуються?**

emits у Vue.js — це механізм для оголошення подій, які компонент може випускати (емітити) та яких можна очікувати від нього. Вони використовуються для взаємодії між компонентами, зокрема, для того, щоб дочірні компоненти могли сповіщати батьківські компоненти про певні події або зміни.

Ось основні аспекти emits у Vue.js:

* Оголошення подій: emits дозволяє явно визначити, які події може випускати компонент. Це робить ваш код більш передбачуваним і документованим.
* Забезпечення коректності: Оголошення подій за допомогою emits допомагає уникнути помилок, забезпечуючи перевірку того, що компонент випускає лише ті події, які були оголошені.
* Взаємодія між компонентами: Дочірні компоненти можуть використовувати emits для сповіщення батьківських компонентів про певні дії або зміни, що дозволяє налаштувати взаємодію між компонентами.

Застосування emits:

* Сповіщення про зміни: Наприклад, коли користувач змінює дані в дочірньому компоненті, ви можете випустити подію, щоб батьківський компонент міг відреагувати на ці зміни.
* Взаємодія між компонентами: Коли один компонент взаємодіє з іншим, події дозволяють зручно передавати дані і сповіщати про події.

Завдяки emits ваші компоненти стають більш гнучкими і легшими у використанні, що сприяє кращій організації коду і зрозумілості взаємодії між компонентами.

**5. Як передати подію з дочірнього компонента до батьківського компонента у Vue?**

Щоб передати подію з дочірнього компонента до батьківського компонента у Vue.js, вам потрібно скористатися механізмом подій (event system). Ось як це зробити:

1. Випуск події у дочірньому компоненті

В дочірньому компоненті ви використовуєте метод this.$emit для випуску події. Метод this.$emit приймає два параметри: ім'я події та необов'язкові дані, які ви хочете передати з подією.

Ось приклад дочірнього компонента, який випускає подію update:

<!-- ChildComponent.vue -->

<template>

<button @click="notifyParent">Click me</button>

</template>

<script>

export default {

methods: {

notifyParent() {

// Випускає подію 'update' і передає дані 'Hello from child!'

this.$emit('update', 'Hello from child!');

}

}

};

</script>

2. Слухання події у батьківському компоненті

В батьківському компоненті ви прослуховуєте подію, використовуючи директиву v-on (або скорочення @). Ви вказуєте ім'я події та метод, який повинен бути викликаний при спрацьовуванні події.

Ось приклад батьківського компонента, який слухає подію update від дочірнього компонента:

<!-- ParentComponent.vue -->

<template>

<ChildComponent @update="handleUpdate" />

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

components: { ChildComponent },

methods: {

handleUpdate(message) {

console.log('Event received with message:', message);

}

}

};

</script>

Важливі моменти:

* Ім'я події: Ім'я події (у цьому випадку update) повинно збігатися між дочірнім і батьківським компонентом.
* Передача даних: Дані, які передаються разом з подією, можна отримати як параметри в методі обробника події у батьківському компоненті.
* Валідація подій: У Vue 3 можна використовувати emits для оголошення подій і валідації їх параметрів.

Цей підхід дозволяє дочірньому компоненту сповіщати батьківський компонент про події або зміни, що робить компоненти більш взаємодійними і гнучкими.

**6. Як визначити події, які компонент може "викидати" (emit)?**

Щоб визначити події, які компонент може "викидати" (emit) у Vue.js, ви можете скористатися властивістю emits. Це дозволяє вам явно оголосити, які події ваш компонент може випускати, і, у Vue 3, ви можете також вказати валідацію для параметрів подій.

Ось як це зробити:

* Оголошення подій у Vue 3

Оголошення подій:

Ви оголошуєте події у властивості emits вашого компонента. Можна просто вказати список подій або додати валідацію для кожної події.

<script>

export default {

emits: ['update', 'delete'], // Простий список подій

methods: {

notifyParent() {

this.$emit('update', 'New data'); // Випуск події

}

}

};

</script>

* Оголошення подій з валідацією параметрів:

Ви також можете задати функції валідації для кожної події, щоб забезпечити правильний формат параметрів:

<script>

export default {

emits: {

update: (payload) => typeof payload === 'string', // Валідація для події 'update'

delete: null // Подія 'delete' не має параметрів або валідації

},

methods: {

notifyParent() {

this.$emit('update', 'New data'); // Випуск події з даними

}

}

};

</script>

* Оголошення подій у Vue 2

У Vue 2 немає прямої підтримки властивості emits. Ви можете документувати події у вашій документації або використовувати коментарі у коді. Випуск подій у Vue 2 також відбувається через this.$emit.

<script>

export default {

methods: {

notifyParent() {

this.$emit('update', 'New data'); // Випуск події

}

}

};

</script>

* У батьківському компоненті ви прослуховуєте ці події, використовуючи директиву v-on або скорочення @.

<template>

<ChildComponent @update="handleUpdate" />

</template>

<script>

export default {

methods: {

handleUpdate(data) {

console.log('Event received with data:', data);

}

}

};

</script>

Важливі моменти:

* Документування подій: Хоча emits у Vue 3 допомагає явно визначати події, у Vue 2 часто використовується документація або коментарі для опису подій, які компоненти можуть викидати.
* Валідація: У Vue 3 ви можете використовувати валідацію для перевірки типів даних, що передаються з подією. Це корисно для забезпечення коректності переданих параметрів.

Цей підхід допомагає робити компоненти більш передбачуваними та полегшує їх використання у великих додатках.

**7. Як можна обробляти події у батьківському компоненті, що надходять від дочірнього компонента?**

Щоб обробляти події у батьківському компоненті, що надходять від дочірнього компонента у Vue.js, слід скористатися механізмом прослуховування подій. Ось покрокове пояснення, як це зробити:

1. Випуск подій у дочірньому компоненті

Спочатку потрібно, щоб дочірній компонент випускав події. Це можна зробити за допомогою методу this.$emit.

ChildComponent.vue

<template>

<button @click="notifyParent">Click me</button>

</template>

<script>

export default {

methods: {

notifyParent() {

// Випускає подію 'update' з деякими даними

this.$emit('update', 'Hello from child!');

}

}

};

</script>

2. Прослуховування подій у батьківському компоненті

У батьківському компоненті потрібно прослуховувати ці події, використовуючи директиву v-on або скорочення @. Вказується ім'я події, яку потрібно слухати, і метод, який повинен бути викликаний при спрацьовуванні події.

ParentComponent.vue

<template>

<ChildComponent @update="handleUpdate" />

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

components: { ChildComponent },

methods: {

handleUpdate(message) {

// Обробляє подію 'update', отримує дані з події

console.log('Event received with message:', message);

}

}

};

</script>

Важливі моменти:

* Ім'я події: Ім'я події (наприклад, 'update' у нашому прикладі) повинно точно збігатися між дочірнім і батьківським компонентом.
* Передача даних: Ви можете передавати дані разом з подією з дочірнього компонента, а у батьківському компоненті ці дані будуть доступні як параметри методу-обробника.

Валідація параметрів подій (Vue 3): Ви можете використовувати властивість emits для валідації параметрів подій у Vue 3, що допомагає забезпечити правильний формат даних.

Валідація параметрів у Vue 3:

<script>

export default {

emits: {

update: (payload) => typeof payload === 'string' // Перевірка типу даних

},

methods: {

notifyParent() {

this.$emit('update', 'Hello from child!');

}

}

};

</script>

**8. Що таке директиви у Vue.js і як вони відрізняються від компонентів?**

Директиви у Vue.js — це спеціальні атрибути, які додають функціональність до HTML елементів, такі як умовне відображення (v-if), цикли (v-for), або двостороння прив'язка (v-model). Вони починаються з префіксу v- і модифікують поведінку або властивості елементів у шаблоні.

Компоненти у Vue.js є більш складними одиницями, які інкапсулюють шаблон, логіку і стилі в єдину структуру. Компоненти можна використовувати повторно для створення складних інтерфейсів.

Основні відмінності:

* Директиви: Модифікують HTML елементи, не маючи власного шаблону.
* Компоненти: Включають власний шаблон, скрипти і стилі, дозволяючи створювати автономні частини інтерфейсу.

**9. Які є вбудовані директиви у Vue.js, і які їхні основні призначення?**

Vue.js має кілька вбудованих директив, кожна з яких має конкретне призначення:

* v-bind: Прив'язує атрибути або пропси до даних. Наприклад, v-bind:href="url".
* v-model: Забезпечує двосторонню прив'язку даних для форм, як-от поля вводу. Наприклад, v-model="inputValue".
* v-if: Умовно відображає елементи в залежності від виразу. Наприклад, v-if="isVisible".
* v-else-if: Використовується разом з v-if для перевірки додаткових умов.
* v-else: Використовується разом з v-if і v-else-if як "інакше".
* v-for: Рендерить список елементів, перебираючи масив. Наприклад, v-for="item in items".
* v-show: Керує видимістю елементів шляхом зміни CSS властивості display. Наприклад, v-show="isVisible".
* v-on: Додає обробники подій до елементів. Наприклад, v-on:click="handleClick".

Ці директиви дозволяють ефективно управляти відображенням і поведінкою елементів у Vue.js.

**10. Як створити власну директиву у Vue.js? Які основні етапи цього процесу?**

Щоб створити власну директиву у Vue.js, виконайте такі основні етапи:

* Оголосіть директиву: Використовуйте Vue.directive для реєстрації нової директиви.

Vue.directive('my-directive', {

// Вказати обробники життєвого циклу

bind(el, binding, vnode) {

// Код для ініціалізації

},

update(el, binding, vnode, oldVnode) {

// Код для оновлення

},

unbind(el, binding, vnode) {

// Код для очищення

}

});

* Використовуйте директиву у шаблоні: Застосуйте свою директиву в шаблоні компонента.

<template>

<div v-my-directive="value">Content</div>

</template>

* Оновлюйте директиву за потреби: Додайте або змініть обробники для управління різними стадіями життєвого циклу елемента.

Ці кроки дозволяють створювати кастомні директиви для специфічної поведінки елементів у Vue.js.