МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Сучасні каркаси web-додатків»

Лабораторна робота № 2

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

Перевірив: Лебідь Максим   
 Володимирович

Запоріжжя  
2024

**Тема**

Створення компонентів з використанням Vue 3 та Tailwind CSS.

**Мета**

1. Навчитися створювати базові компоненти у Vue 3.

2. Опанувати стилізацію компонентів за допомогою Tailwind CSS.

3. Практикувати передачу пропсів та обробку подій між компонентами.

**Засоби та інструменти:**

* Vue 3
* Tailwind CSS
* Node.js (для встановлення залежностей)
* npm або yarn для управління пакетамиЗавдання

**Завдання**

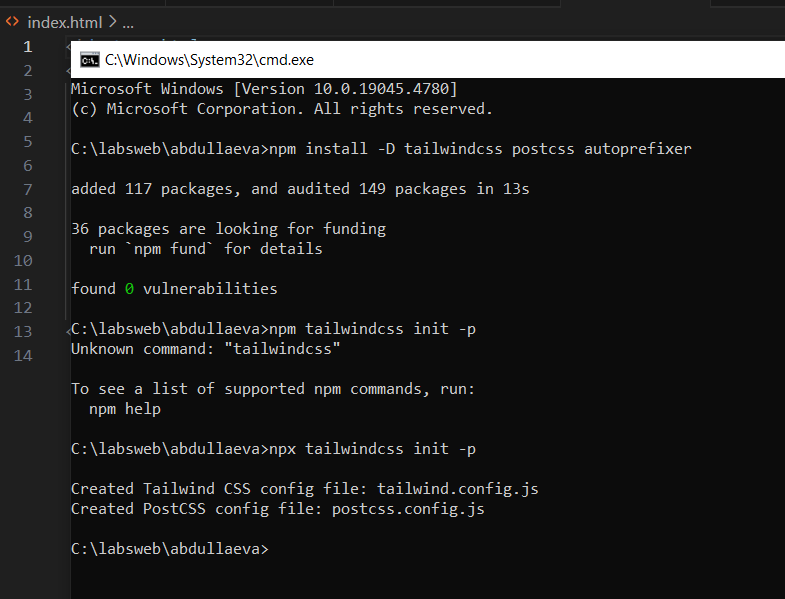
1. Налаштування проекту Vue 3 з Tailwind CSS на базі проекту з попередньої лабораторної роботи.

#Встановіть та налаштуйте Tailwind CSS відповідно до офіційної документації:

npm install -D tailwindcss postcss autoprefixer

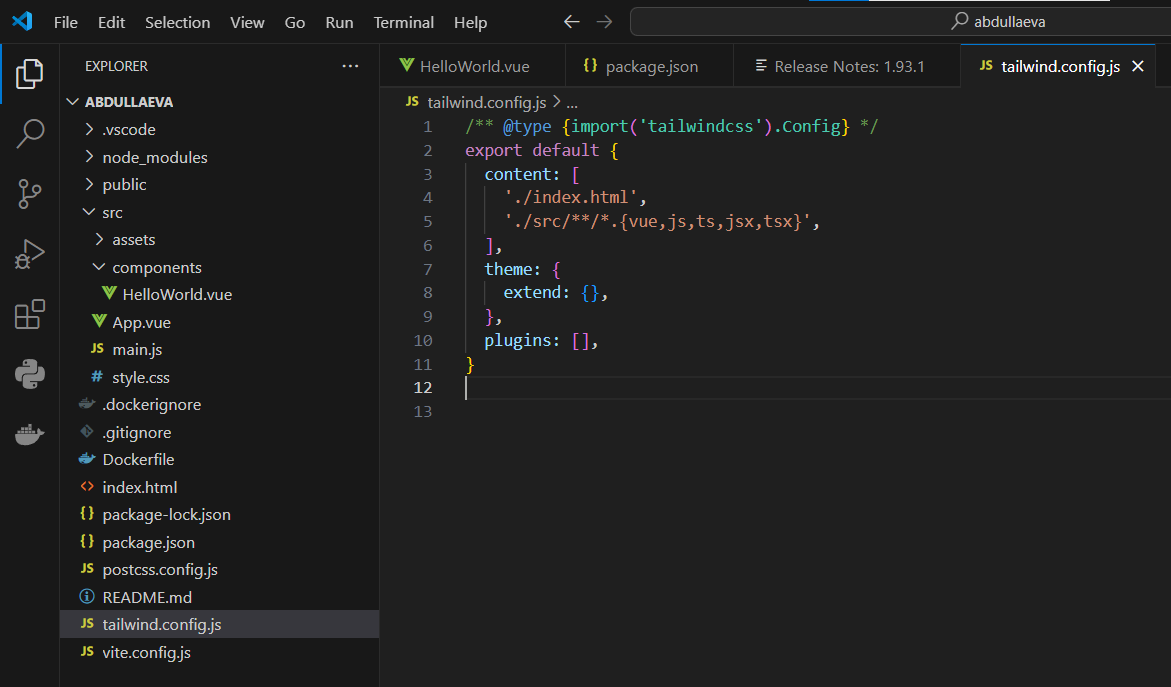
npx tailwindcss init -p

Встановлюємо та ініціалізуємо Tailwind:

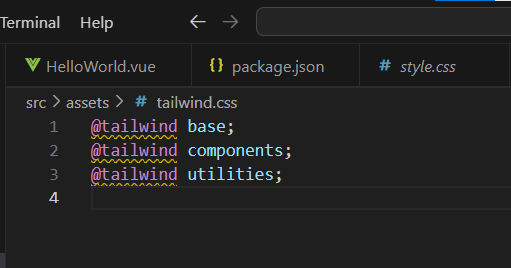


2. Налаштуємо конфігурацію Tailwind:

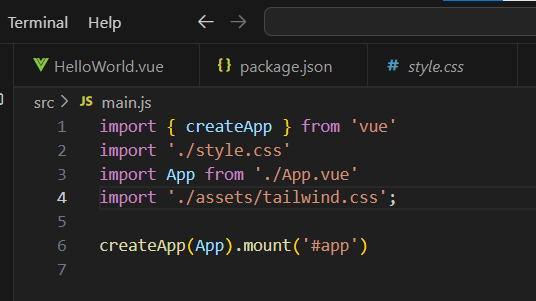
Відкриємо файл tailwind.config.js і додамо шляхи до ваших Vue-файлів у секцію content, щоб Tailwind міг сканувати файли на наявність класів:



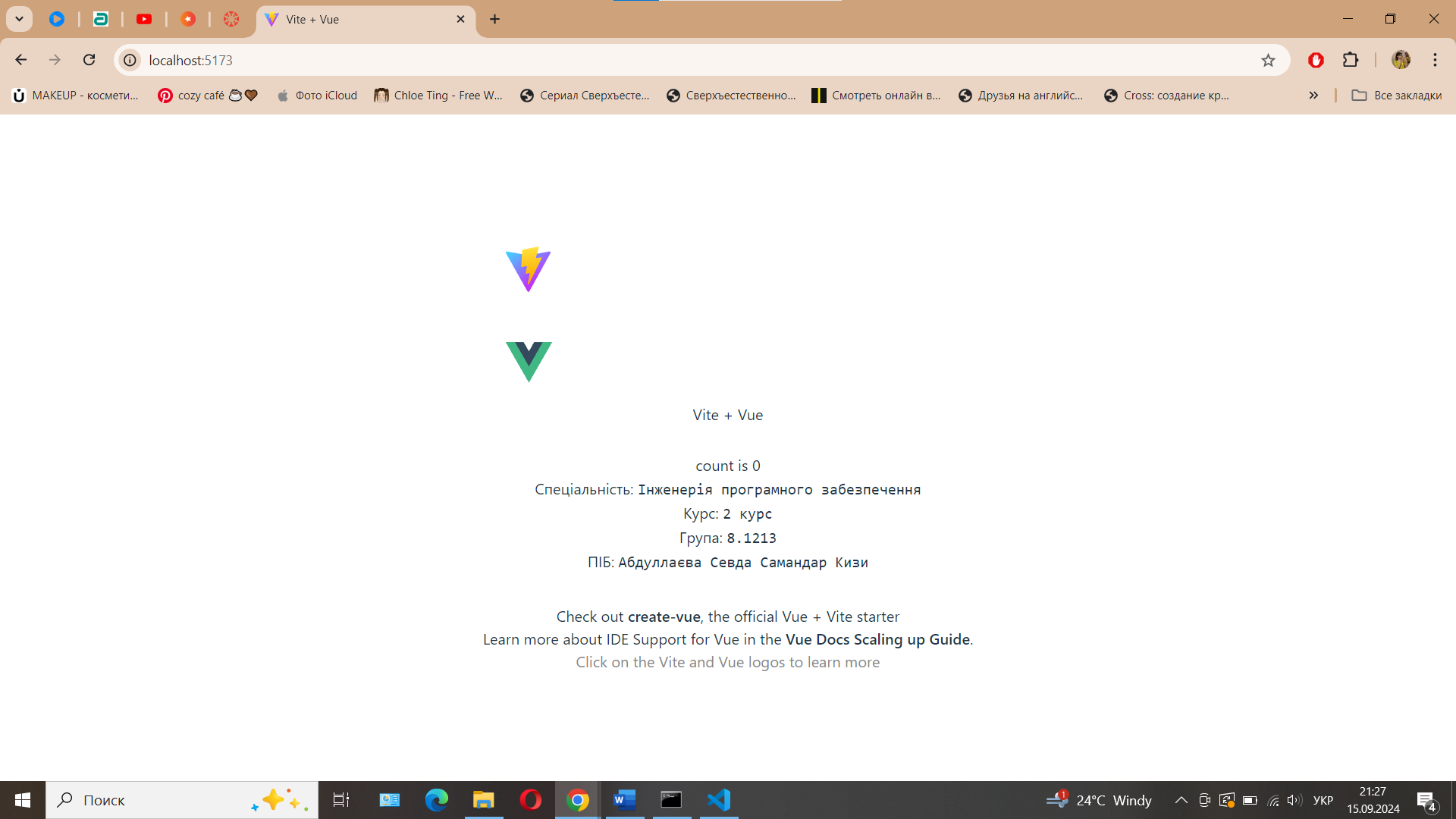
У кореневій папці src створимо файл src/assets/tailwind.css. Додамо до нього наступні директиви:



Імпортуємо Tailwind у проект:



Запустимо проект:



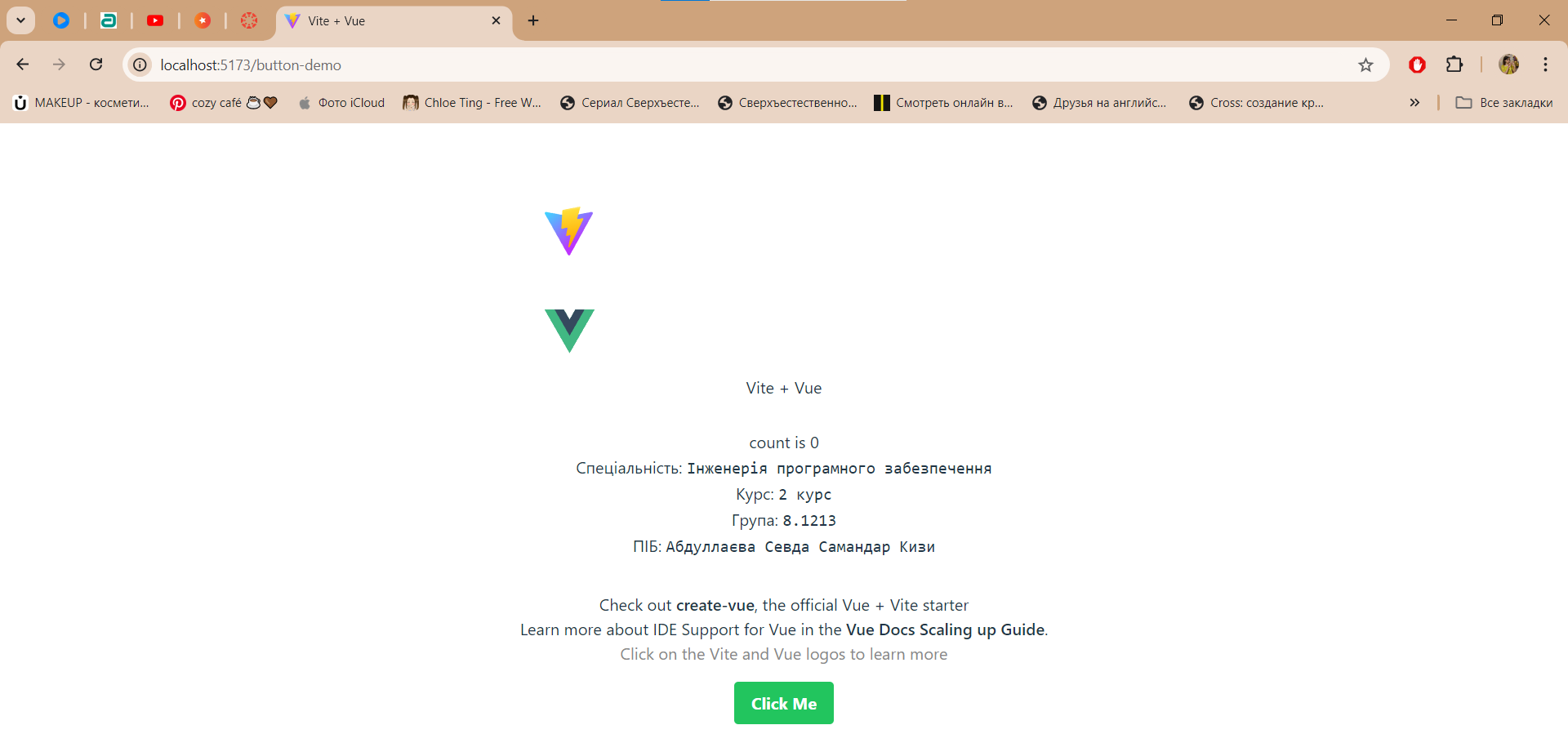
3. Створіть компонент Button за наступними функціональними вимогами:

- 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).

- Використайте Tailwind для стилізації кнопки з урахуванням пропсів.

- Додайте можливість обробки події click на кнопці.



Бачимо, що на сторінці додалась кнопка «Click Me», з урахуванням всіх поставлених задач.



4. Створіть компонент SearchableDropdown за наступними функціональними вимогами:

- 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:

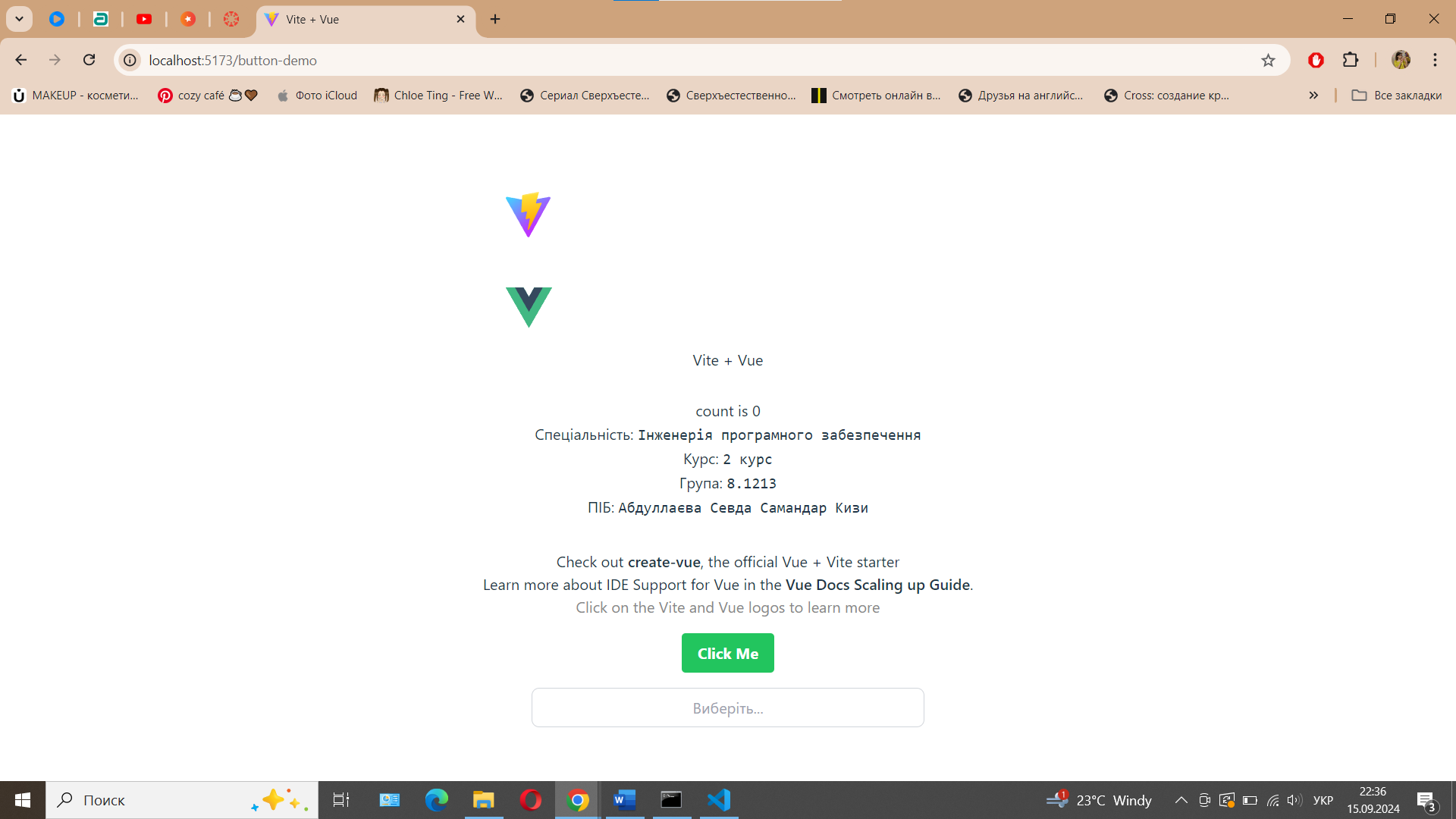
* Компонент підтримує навігацію за допомогою клавіш стрілок та Enter для вибору елементів.

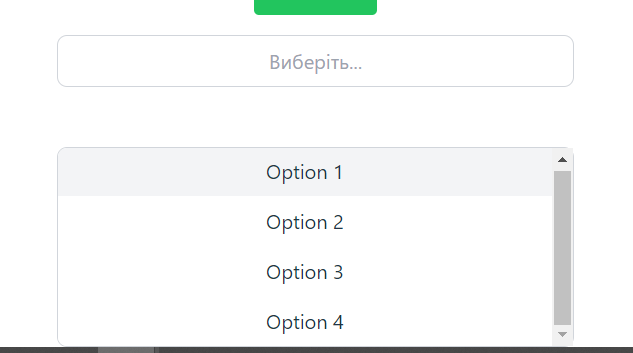
- Скидання вибору (Clear Selection):

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

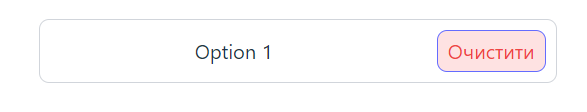
- Кастомізація відображення варіантів:

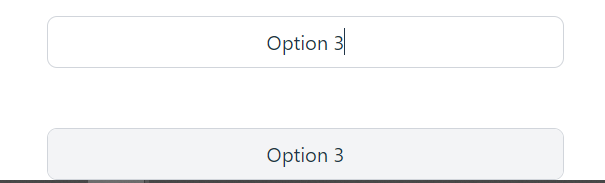
* Користувач може кастомізувати відображення елементів списку (наприклад, додавати іконки).











5. Створіть сторінку з демо, яка відображатиме всі можливі комбінації параметрів для кожного з компонентів:

* Для компонента Button – різні кольори, розміри та наявність іконок.
* Для SearchableDropdown – різні набори даних, фільтрація, можливість вибору кількох елементів.

6. Розмістіть ваш проект в директорії lab2 публічного репозиторію GitHub/GitLab/Bitbucket з назвою <vendor>/mjsf-mag та додайте посилання на репозиторій у звіт.

**Посилання на директорію:** <https://github.com/SevdaAbdullaeva/Modern-frames-of-web-applications.git>

**Контрольні питання**

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

У Vue.js, props – це механізм для передачі даних від батьківського компонента до дочірнього. Це дозволяє батьківському компоненту передавати значення, такі як рядки, числа, об'єкти чи масиви, до дочірніх компонентів. Props використовуються для того, щоб зробити компоненти більш гнучкими та повторно використовуваними, оскільки дозволяють конфігурувати їх ззовні, передаючи дані через атрибути. Це забезпечує чисту та зрозумілу структуру компонентів, де кожен компонент може отримувати необхідні йому дані від батьківського компонента без необхідності звертатися до глобального стану чи змінних.

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

У Vue.js можна використовувати різні типи даних для пропсів, такі як рядки, числа, об'єкти, масиви, булеві значення, функції та навіть інші компоненти. Щоб визначити тип пропсу, ви можете використовувати властивість type в об'єкті props, що вказує, який тип даних очікується. Наприклад, якщо ви очікуєте рядок, ви можете визначити це так: props: { message: String }. Якщо ви хочете, щоб пропс міг бути кількома типами, ви можете передати масив типів, наприклад, props: { value: [String, Number] }. Ви також можете визначити, чи є пропс обов'язковим, використовуючи властивість required, і надати значення за замовчуванням через default.

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

У Vue.js, щоб задати значення за замовчуванням для пропса, ви використовуєте властивість default в об'єкті props. Вона дозволяє визначити значення, яке буде використовуватися, якщо пропс не буде передано батьківським компонентом. Наприклад, у наступному коді пропс message має значення за замовчуванням "Hello World":

<script>

export default {

props: {

message: {

type: String,

default: 'Hello World'

}

}

}

</script>

Значення за замовчуванням може бути будь-якого типу, відповідно до типу пропса, і може бути визначене як просте значення, функція або об'єкт. Якщо значення є об'єктом або масивом, краще використовувати функцію для повернення значення за замовчуванням, щоб уникнути спільного використання одного і того ж об'єкта в кількох екземплярах компонента.

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

У Vue.js, emits – це механізм для визначення імен подій, які компонент може викидати (емітити) і які батьківський компонент може слухати. Це дозволяє дочірнім компонентам сповіщати батьків про різні події або зміни стану. emits використовується для декларативного визначення подій, які компонент може викликати, що покращує читабельність коду та забезпечує більшу ясність щодо того, які події можуть бути згенеровані компонентом.

Щоб задати події, які компонент може викидати, ви використовуєте властивість emits в об'єкті компонента. Наприклад:

<script>

export default {

emits: ['submit', 'cancel'],

methods: {

handleSubmit() {

this.$emit('submit', this.formData);

},

handleCancel() {

this.$emit('cancel');

}

}

}

</script>

У цьому прикладі, компонент може викидати події submit і cancel. Це дозволяє батьківському компоненту слухати ці події та реагувати на них, наприклад:

<template>

<ChildComponent @submit="handleFormSubmit" @cancel="handleFormCancel" />

</template>

<script>

export default {

methods: {

handleFormSubmit(data) {

console.log('Form submitted with data:', data);

},

handleFormCancel() {

console.log('Form cancelled');

}

}

}

</script>

Використання emits допомагає в управлінні подіями та забезпечує чітке визначення того, які події можуть бути викликані дочірніми 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, ви використовуєте метод this.$emit в дочірньому компоненті для викиду події. Це дозволяє дочірньому компоненту сповіщати батьківський компонент про певні дії або зміни. У батьківському компоненті ви можете слухати ці події за допомогою директиви v-on або її короткої форми @.

Ось як це працює:

У дочірньому компоненті ви використовуєте this.$emit для викиду події. Наприклад, якщо ви хочете викинути подію submit разом з даними форми, ви пишете:

<script>

export default {

methods: {

handleSubmit() {

this.$emit('submit', this.formData);

}

}

}

</script>

-У батьківському компоненті ви прослуховуєте цю подію, використовуючи директиву v-on або коротку форму @ на компоненті. Наприклад, щоб прослуховувати подію submit і викликати метод handleFormSubmit при її виникненні, ви пишете:

<template>

<ChildComponent @submit="handleFormSubmit" />

</template>

<script>

export default {

methods: {

handleFormSubmit(data) {

console.log('Form submitted with data:', data);

}

}

}

</script>

У цьому прикладі, коли дочірній компонент викине подію submit, метод handleFormSubmit батьківського компонента буде викликаний з переданими даними. Це дозволяє здійснювати комунікацію між компонентами та обробляти події в батьківському компоненті.

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

Щоб визначити події, які компонент може "викидати" у Vue.js, ви використовуєте властивість emits в об'єкті компонента. Вона дозволяє явно вказати, які події можуть бути згенеровані компонентом, що покращує читабельність коду та забезпечує чіткість у тому, які події компонент може відправляти.

Ось як це можна зробити:

- У компоненті, в якому ви будете викидати події, додайте властивість emits до об'єкта експорту. В emits ви можете перерахувати всі події, які можуть бути викинуті. Це може бути масив рядків або об'єкт, що дозволяє також вказати, які параметри можуть бути передані разом з подією.

<script>

export default {

emits: ['submit', 'cancel'],

methods: {

handleSubmit() {

this.$emit('submit', this.formData);

},

handleCancel() {

this.$emit('cancel');

}

}

}

</script>

- В цьому прикладі, компонент оголошує, що він може викидати події submit і cancel. Коли в компоненті викликається this.$emit('submit', this.formData), батьківський компонент може слухати подію submit і обробляти дані, які передаються разом з нею.

Цей підхід допомагає підтримувати код чистим і зрозумілим, оскільки всі можливі події, які компонент може викидати, явно вказані, і це полегшує розуміння того, як компонент взаємодіє з іншими частинами програми.

**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, вам потрібно прослуховувати ці події у батьківському компоненті за допомогою директиви v-on або її короткої форми @. У дочірньому компоненті ви використовуєте this.$emit для викиду подій, а в батьківському компоненті – для прослуховування та обробки цих подій.

Ось як це робиться:

- У дочірньому компоненті, де ви хочете викидати події, використовуйте this.$emit у відповідному методі або дії. Наприклад, якщо ви хочете викинути подію submit при натисканні на кнопку, ваш код може виглядати так:

<script>

export default {

methods: {

handleSubmit() {

this.$emit('submit', this.formData);

}

}

}

</script>

<template>

<button @click="handleSubmit">Submit</button>

</template>

- У батьківському компоненті прослуховуйте події, що надходять від дочірнього компонента, використовуючи директиву v-on або її коротку форму @ у шаблоні. Ви вказуєте ім'я події, яку ви хочете прослуховувати, і метод, який має бути викликаний при її виникненні. Наприклад, для прослуховування події submit ви можете написати так:

<template>

<ChildComponent @submit="handleFormSubmit" />

</template>

<script>

export default {

methods: {

handleFormSubmit(data) {

console.log('Form submitted with data:', data);

}

}

}

</script>

У цьому прикладі, коли дочірній компонент викине подію submit, метод handleFormSubmit батьківського компонента буде викликаний, і дані, що передаються з подією, будуть доступні для обробки. Це дозволяє батьківському компоненту реагувати на події, що виникають у дочірньому компоненті, і виконувати необхідні дії.

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

У Vue.js директиви – це спеціальні атрибути, які додаються до елементів у шаблонах для надання їм особливого функціоналу. Директиви зазвичай використовуються для роботи з DOM, такими як виведення даних, обробка подій або умовне рендеринг. Наприклад, директива v-if використовується для умовного рендерингу елементів, а директива v-bind дозволяє зв'язати атрибути елемента з даними компонента.

Компоненти ж у Vue.js – це самостійні, повторно використовувані блоки коду, які мають свій власний шаблон, логіку та стиль. Компоненти можуть містити власні директиви, дані, методи і навіть інші компоненти. Вони дозволяють розбивати інтерфейс на менші, управлінські частини, що полегшує підтримку та повторне використання коду.

Основна відмінність між директивами та компонентами полягає в тому, що директиви переважно відповідають за маніпуляцію DOM-елементами та атрибутами, тоді як компоненти є більш складними одиницями, які включають шаблони, логіку та стилі, і можуть бути самостійними частинами інтерфейсу. Директиви надають функціональність на рівні елемента, тоді як компоненти забезпечують більш високий рівень абстракції та структурованості.

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

У Vue.js є кілька вбудованих директив, кожна з яких має своє специфічне призначення для роботи з DOM. Основні вбудовані директиви включають:

* v-bind – ця директива використовується для зв'язування атрибутів DOM з даними компоненту. Вона дозволяє динамічно змінювати атрибути елемента, такі як href, class, style і інші. Наприклад, v-bind:href="url" прив'язує атрибут href до змінної url.
* v-model – ця директива забезпечує двосторонній зв'язок між даними компоненту та елементом форми. Вона автоматично оновлює дані при зміні значення в елементі форми і навпаки. Використовується для елементів форм, таких як <input>, <textarea>, <select>.
* v-for – ця директива використовується для рендерингу списків даних. Вона ітерує через масив або об'єкт та створює елементи для кожного з його елементів. Наприклад, v-for="item in items" створює елемент для кожного item в масиві items.
* v-if – ця директива дозволяє умовно рендерити елементи. Якщо умова, задана в v-if, істинна, елемент буде вставлений в DOM. Якщо умова хибна, елемент буде видалений з DOM.
* v-else-if і v-else – ці директиви працюють разом з v-if, щоб реалізувати альтернативні гілки умовного рендерингу. v-else-if перевіряє іншу умову, якщо попередня умова v-if не була істинною, а v-else виконує код, якщо жодна з попередніх умов не виконалась.
* v-show – ця директива використовується для умовного відображення елементів. Вона схожа на v-if, але замість того, щоб додавати або видаляти елементи з DOM, вона просто змінює CSS-властивість display, що дозволяє елементу залишатися в DOM, але бути невидимим.
* v-on – ця директива використовується для обробки подій DOM, таких як кліки, ввід тексту тощо. Вона дозволяє прив'язувати методи компонента до подій елемента. Наприклад, v-on:click="handleClick" викликає метод handleClick, коли на елемент натискають.
* v-pre – ця директива інструктує Vue пропустити компіляцію для даного блоку, що може бути корисно для оптимізації або для виведення необробленого шаблонного коду.
* v-cloak – ця директива використовується для приховування елементів, поки Vue не завершить компіляцію шаблону. Це запобігає відображенню необробленого шаблонного коду на момент завантаження сторінки.
* v-once – ця директива рендерить елемент або компонент тільки один раз і не оновлює його, навіть якщо дані змінюються.

Ці директиви допомагають управляти рендерингом, взаємодією з користувачем і динамічним оновленням елементів у Vue.js.

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

Щоб створити власну директиву у Vue.js, вам потрібно виконати кілька основних етапів. Це дозволяє вам додавати спеціальну функціональність до елементів у вашому додатку.

Ось основні етапи створення власної директиви у Vue.js:

- Оголошення директиви: Спочатку потрібно оголосити вашу директиву. У Vue.js це можна зробити глобально або локально. Глобальні директиви реєструються за допомогою Vue.directive, а локальні – всередині компонента. Наприклад, щоб створити глобальну директиву, ви можете використати:

Vue.directive('my-directive', {

// Опції директиви

});

Для локальної директиви всередині компонента:

javascript

Копировать код

export default {

directives: {

'my-directive': {

// Опції директиви

}

}

}

- Визначення опцій директиви: Директива повинна містити набір методів або опцій, які визначають її поведінку. Основні методи включають:

* bind: викликається один раз, коли директива прив'язується до елемента. Ідеально підходить для ініціалізації.
* inserted: викликається після того, як елемент вставлений в DOM. Може використовуватися для роботи з DOM після того, як він був створений.
* update: викликається при оновленні значення директиви. Може використовуватися для реагування на зміни в даних.
* componentUpdated: викликається після оновлення компоненту, що використовує директиву.
* unbind: викликається один раз, коли директива відв'язується від елемента. Використовується для очищення або скидання.

Наприклад:

Vue.directive('my-directive', {

bind(el, binding, vnode) {

el.style.color = binding.value;

}

});

- Використання директиви: Після того, як ви оголосили і визначили директиву, ви можете використовувати її у вашому шаблоні. Якщо директива глобальна, її можна використовувати в будь-якому компоненті. Якщо локальна, то тільки в компоненті, де вона була зареєстрована.

<div v-my-directive="'red'">This text will be red.</div>

- Тестування та вдосконалення: Перевірте свою директиву, щоб переконатися, що вона працює як очікується. Внесіть необхідні корективи для поліпшення її функціональності або сумісності.

Створення власних директив у Vue.js дозволяє вам розширити можливості вашого додатку, додаючи спеціалізовану поведінку до елементів DOM відповідно до ваших потреб.