Building Real-World Web Applications with Vue.js 3

. . . .

Потреба у Vue.js

Існує низка інтерфейсних фреймворків, і це постійно розвивається та змінюється ландшафт. На даний момент екосистема Vue.js є дуже зрілим ландшафтом, який пропонує численні плагіни та інструменти, створені на його основі.

Vue.js — це проект із відкритим кодом, керований спільнотою, який підтримується та розробляється людьми з усього світу. Він виник як особистий побічний проект Евана Ю і розрісся до фреймворку, який широко прийняли всі типи організацій, такі як NASA, Apple, Google і Microsoft. Оскільки Vue.js фінансується спонсорами обох компаній як окремих осіб, це незалежна структура.

Наразі Vue.js знаходиться у версії 3, що є суттєвою зміною порівняно з попередньою версією, хоча більшість шаблонів все ще застосовуються. Vue.js 3 підтримує всі браузери, сумісні з ES2015.

Давайте розглянемо кілька причин, чому варто вибрати Vue.js для створення веб-додатків:

• Він ефективний, тому що створений для оптимізації з нуля.

• Він легкий, його можна потрясти і містить лише код, необхідний для запуску програми. Мінімальний розмір коду (після оптимізації на етапі збірки) становить близько 16 КБ.

• Він має високу масштабованість, використовує бажані шаблони, такі як Single File Components і Composition API, що робить його придатним для корпоративних програм.

Single File Components є частиною філософії Vue.js, де шаблон, сценарій і стиль компонента інкапсульовані в одному файлі з метою покращення організації, читабельності та зручності обслуговування коду.

Composition API дозволяє краще організувати та повторно використовувати код у всій програмі, що робить код більш модульним і простим у обслуговуванні.

Крім усіх цих переваг, крива навчання дуже доступна для розробників початкового рівня. Завдяки синтаксису, схожому на класичні нотації HTML, JavaScript і CSS, легко розпочати роботу та зорієнтуватися.

У цьому розділі ми проведемо вас через початкове налаштування та пройдемо кроки та налаштування, які ви можете використовувати як шаблон для всіх майбутніх проектів Vue.js. Ми застосуємо рекомендовані налаштування, щоб переконатися, що ви навчитеся та застосовуєте найкращі практики з самого початку.

Спершу ми переконаємося, що у вас налаштовано середовище розробника, щоб ви могли почати створювати інтерактивні веб-додатки!

Вимоги та інструменти

Щоб ефективно розпочати розробку Vue.js, нам потрібно буде переконатися, що ви дійсно можете запускати та розумно редагувати код. Хоча технічно ви можете запустити код за допомогою бібліотеки з мережі доставки вмісту (CDN), це не рекомендується для реальних програм. Як також зазначено в офіційних документах (https://vuejs.org/guide/introduction.html), налаштування збірки не вимагається, а недоліком є те, що це налаштування не підтримує синтаксис компонента з одним файлом і залишає вам мало контролю. над оптимізацією програм, як-от компіляція, мініфікація та відкладене завантаження.

У цій книзі ми будемо використовувати пакет Vue.js npm, а потім використовуватимемо його для розробки початкових проектів, на основі яких можна буде працювати. Ми будемо запускати всі наші проекти за допомогою командного рядка. Щоб використовувати пакет npm, вам потрібно буде встановити Node.js (середовище виконання JavaScript). Обов’язково встановіть принаймні Node.js версії 18. npm — це загальнодоступне сховище (https://www.npmjs.com/), де розробники публікують, діляться та використовують пакети JavaScript.

Інтернет-ресурси

Vue.js надає майданчик для онлайн-розробки, https://sfc.vuejs.org/ , але я хотів би відзначити StackBlitz (https://stackblitz.com/), де ви можете створювати повні середовища розробки, які працюють у браузер. Хоча це не корисно для розгортання програм, це чудовий спосіб перевірити доказ концепції або просто використовувати його як невеликий ігровий майданчик.

Ви можете просто зареєструватися, почати новий проект і вибрати шаблон Vue.js 3, щоб розпочати роботу. Приклади коду будуть доступні на GitHub, де ви можете клонувати або розділити репозиторій, щоб перевірити свій власний код на робочий приклад.

Для подальшого використання документи Vue.js (https://vuejs.org/guide/introduction.html) дуже доступні та пропонують покрокове пояснення всіх можливих контекстів. Я, звичайно, рекомендую перевірити їх, щоб глибше зрозуміти теми, які ми будемо висвітлювати.

Налаштування середовища розробки

Існує багато способів написання та редагування коду, і з часом ви знайдете потік, який вам найкраще підходить. У цій книзі ми почнемо із загально рекомендованих налаштувань. Не соромтеся вносити зміни, які вам підходять.

Розробка Vue.js відбувається в середовищі, яке дозволяє вам ефективно писати код, виділяючи правильний код і допомагаючи вам виявляти помилки, перш ніж зберегти зміни. Налагодження та перевірка коду можуть відбуватися на різних етапах, але в рамках цієї книги ми будемо використовувати інтерфейс розробки, а також середовище браузера.

Почнемо зі встановлення широко поширеного інтерфейсу розробки.

Інтегроване середовище розробки

Інтегроване середовище розробки (IDE) допомагає вам писати та редагувати код, підтримуючи такі допоміжні засоби, як підсвічування синтаксису, форматування та плагіни, пов’язані з обраною структурою. Для цього підійде будь-який сучасний редактор, але в цій книзі ми будемо використовувати Microsoft Visual Studio Code (VSCode), який є безкоштовним для використання та забезпечує гарний досвід розробника; його можна завантажити з https://code.visualstudio.com/ .

Окрім інсталяції IDE, я рекомендую наступні плагіни, які роблять роботу розробника набагато приємнішою:

• **Vue Language Features (Volar**): підтримує розмітку фрагментів Vue.js 3 і виділення

• **Vue Volar extension pack**: додає деякі рекомендовані плагіни, щоб допомогти автоматизувати деякі роботи під час кодування

• **Better comments**: для кращої розмітки коментарів у коді

• **Indent-rainbow**: застосовує колір до блоків коду з відступами, щоб швидко визначити рівні відступів

Vue.js можна розробляти за допомогою багатьох інших IDE, таких як WebStorm, Sublime Text, Vim/NeoVim і Emacs. Виберіть те, що вам підходить, майте на увазі, що скріншоти будуть показані з використанням рекомендованих налаштувань VSCode, як описано раніше.

Мій перший додаток

Давайте перевіримо наші набуті інструменти та знання, створивши нашу першу програму Vue.js, чи не так?

Зазвичай ви починаєте з відкриття свого CLI та переходу до папки, з якої ви хочете розпочати свої проекти. Введення наступної команди створює новий порожній проект за допомогою офіційного інструменту createvue:

```
npm init vue@latest
```

Натисніть у, щоб продовжити, виберіть **my-first-vue** як назву проекту та виберіть параметри, показані на наступному малюнку:

Малюнок 1.1 – Використання Vue CLI для розробки програми з попередніми налаштуваннями

Ми вибрали TypeScript як надмножину JavaScript, яка додає статичний тип. Ми також увімкнули ESLint і Prettier. ESLint перевіряє синтаксичні помилки, проблеми форматування та невідповідності стилю коду, і навіть може інтегруватися з вашим IDE, щоб візуально позначати проблемний код. Prettier використовується для забезпечення узгодженого стилю коду. Ці три варіанти покращують роботу розробника, висвітлюючи потенційні проблеми перед запуском коду.

Потім, дотримуючись інструкцій, ви можете ***перейти до створеної папки*** та ввести ***npm install***, щоб установити необхідні залежності. Це завантажить необхідні файли пакетів із реєстру npm і встановить їх у підпапку node_modules вашого проекту.

Якщо ви запустите ***npm run dev***, проект запустить сервер розробки, до якого ви зможете отримати доступ за допомогою свого браузера. Зазвичай локальна адреса буде схожа на http://127.0.0.1:5173/.

Якщо ви відкриєте цю URL-адресу в браузері, ви побачите свою першу програму Vue.js! За замовчуванням це порожній початковий файл, який містить багато покажчиків і посилань, які ми вже розглянули на цьому етапі, але це хороша відправна точка для будь-якого розробника, починаючи з Vue.js.

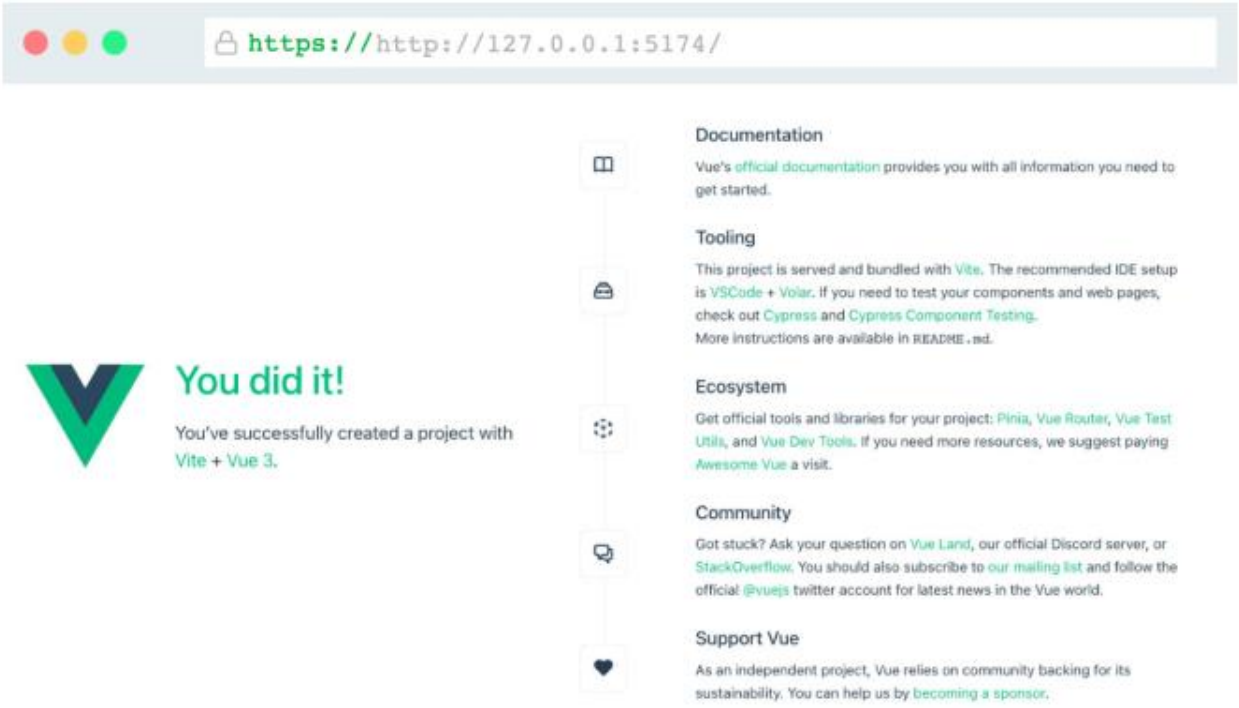

Рисунок 1.2 – Ваша перша програма Vue.js!

Після успішного встановлення ми можемо ближче подивитися на те, що насправді було встановлено. Давайте зануримося в інсталяційні файли!

Проект в IDE

Тепер, якщо ви відкриєте проект у вибраному середовищі IDE, ви помітите попередньо визначену структуру. Це буде застосовно до всіх проектів, які побудовані таким чином. Давайте швидко розглянемо структуру:

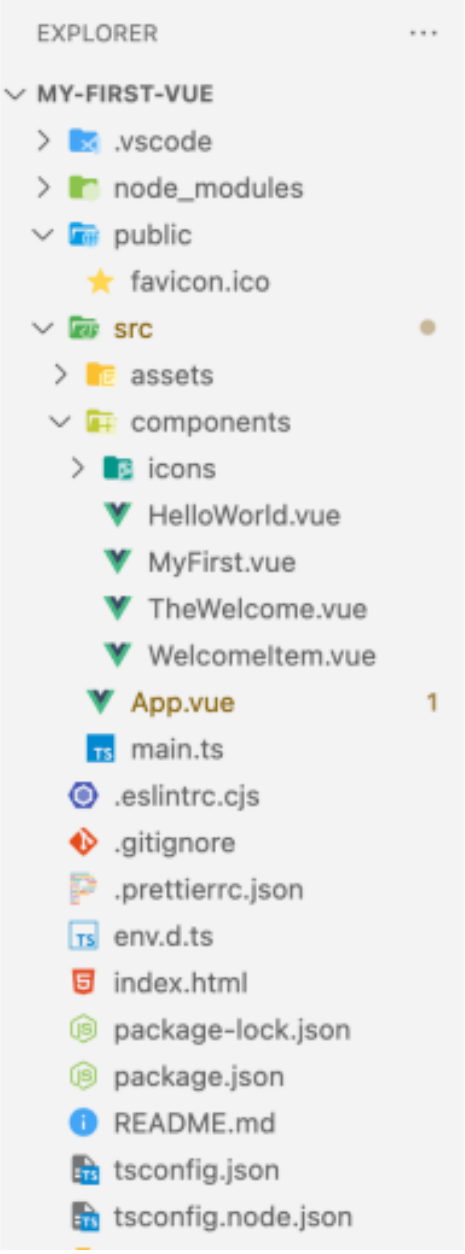

Рисунок 1.3 – Розгорнута структура папок початкової програми

У корені проекту ви знайдете різні типи файлів, які є специфічними для налаштування проекту. Основними файлами тут ***є index.html*** і ***package.json***. Файл index.html є точкою входу програми. Це легкий шаблон HTML з елементом **div**, який має ідентифікатор програми (**id app**), яка буде точкою монтування програми.

Файл ***package.json*** — це файл, який описує проект як пакет, визначає сценарії вузлів, які можна виконати, а також містить посилання на всі пакети,

від яких залежить проект. Папка ***node_modules*** — це папка, яка містить усі встановлені пакети з файлу package.json. Для наших цілей цю папку можна вважати лише для читання.

Потім у нас є папки ***public*** і ***src***. Спільна папка містить статичні ресурси, такі як шрифти, зображення та значки, які не повинні оброблятися системою збірки. У початковому проекті ви знайдете favicon.ico за замовчуванням.

Нарешті, папка ***src*** (скорочення від source) — це папка, до якої ми будемо робити найбільше змін. На даний момент вона містить два кореневих файли. Файл ***main.ts*** реєструє програму Vue і стилі та монтує їх до шаблону HTML.

Файл ***App.vue*** є точкою входу програми Vue.js. Якщо ви відкриєте його, ви можете знайти деякі знайомі синтаксиси, змішані в одному файлі, як-от теги сценарію, HTML і CSS. Ми дійдемо до цього пізніше.

Він також містить папку ***assets*** (активів), подібну до загальнодоступної папки, з тією різницею, що ці папки можуть і будуть оброблені системою збирання. І, нарешті, є папка ***components***, куди можна розмістити компоненти, з яких складається програма. Якщо ви використовуєте компоненти одного файла, кожен виконуватиме певну роль і інкапсулюватиме шаблон, сценарій і стилі. Ви вже можете побачити кілька компонентів, які складають початкову сторінку за замовчуванням.

Ваші перші кроки кодування

Давайте створимо перший компонент і додамо його до програми:

1. Створіть новий файл у папці компонентів під назвою MyFirst.vue.

Компонент Vue.js бажано називати принаймні двома словами з верблюжого регістру та зазвичай складається зі ***сценарію, шаблону та блоку стилю*** (*script, template, and style block*). Жоден із них не є обов'язковим (хоча блок стилю без контексту мало б цінності).

2. Давайте створимо невеликий фрагмент HTML:

```
<template>
    <div>My first <span>Vue.js</span> component!</div>
</template>
```

3. У App.vue ви вже можете використовувати це як компонент Vue.js! Якщо ви відкриєте App.vue, ви побачите тег сценарію з операторами імпорту. Ви можете видалити рядок імпорту TheWelcome і замінити його таким:

```
import MyFirst from './components/MyFirst.vue'
```

4. Далі в тегу шаблону ви можете видалити HTML-тег <TheWelcome /> і замінити його HTML-нотацією <MyFirst />.

Якби ви все ще запускали код, ви б помітили, що браузер оновився, щоб відобразити зміни. Це називається гарячим перезавантаженням і забезпечує плавний потік розробки. Якщо ви зупинили процес, ви можете перезапустити його та знову переглянути сторінку в браузері.

Ви повинні побачити компонент, який ви створили!

5. Давайте додамо блок стилів, щоб додати CSS до компонента та побачити гаряче перезавантаження в дії. У файлі MyFirst.vue додайте наступний код під блоком шаблону:

```
<style scoped>
div {
  color: #35495f;
  font-size: 1.6rem;
}
span {
  color: #41b883;
  font-weight: 700;
}
</style>
```

Вміст блоку стилів буде оброблено як звичайний файл CSS. Атрибут scoped означає, що визначення стилю div і span охоплюють лише цей компонент. Vue додає унікальний атрибут даних до віртуальної DOM і приєднує правила CSS до цього атрибута. У App.vue ви можете побачити, що глобальні стилі також підтримуються.

У браузері ви побачите, що компонент оновлюється з новим стилем! Тепер, коли ми знайомі із середовищем розробки, ми почнемо створювати більш інтерактивний компонент у наступному розділі.

Створення програми Todo List

Тепер, коли у нас налаштовано середовище розробки, ми розпочнемо написання невеликої першої програми. У цьому розділі ми створимо програму Todo list, яка навчить нас, як працює реактивність Vue.js і віртуального DOM. Ви можете використовувати програму Todo list як посібник для відстеження прогресу в цій книзі!

Давайте складемо це завдання з деякими практичними вимогами:

• Ми подбаємо про те, щоб ви бачили список предметів

• Кожен елемент матиме прапорець

• Список буде відсортовано спочатку за непозначеними елементами, а потім за позначеними

• Статус елемента має зберігатися браузером під час майбутніх відвідувань

Існують різні способи написання дійсного компонента Vue.js. Наразі Composition API надається перевагу над Options API. Options API використовує об'єктно-орієнтований підхід, тоді як Composition API дозволяє використовувати більш багаторазовий спосіб написання та організації коду.

У цій книзі ми будемо використовувати нотацію Composition API зі скороченням для функції налаштування, якщо не зазначено інше. Такий спосіб написання коду усуває багато шуму та повторюваних операцій із ваших компонентів і є дуже ефективним способом роботи. Ми також будемо використовувати варіант TypeScript, оскільки він підтримується з коробки та пропонує кращий Developer eXperience (DX) завдяки суворій типізації.

Примітка

Ви можете прочитати більше про синтаксис тут: https://vuejs.org/api/sfc-scriptsetup.html#script-setup . Докладніше про визначення компонентів за допомогою TypeScript можна знайти тут: https://vuejs.org/guide/typescript/composition-api.html#using-script-setup.

У цьому розділі ми розглянемо такі теми:

• Використання інструменту CLI для створення спеціального середовища для програми

• Концепція реактивності Vue.js

• Стилізація за допомогою CSS

• Перший погляд на Vue.js DevTools

У нас немає технічних вимог, окрім тих, які ми розглянули в попередньому розділі, тож ми можемо розпочати роботу одразу!

Новий проект

Давайте почнемо зі створення нового проекту, використовуючи команди CLI з попереднього розділу. Відкрийте вікно терміналу в папці проектів і дотримуйтеся таких інструкцій:

```
npm init vue@latest
```

Натисніть y, щоб продовжити, використовуйте vue-todo-list як назву проекту та виберіть параметри, показані на наступному знімку екрана:

```
Vue.js - The Progressive JavaScript Framework

✔ Project name: … vue-todo-list
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add an End-to-End Testing Solution? › No
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in /Users/jquinten/Projects/packt/vue-todo-list...

Done. Now run:

  cd vue-todo-list
  npm install
  npm run format
  npm run dev
```

Рисунок 2.1 – Конфігурація налаштування програми Todo list

Дотримуйтеся наведених інструкцій, щоб встановити залежності та відкрийте свою улюблену IDE, щоб почати.

Очищення інсталяції за замовчуванням

Давайте спочатку очистимо папку компонентів, видаливши HelloWorld.vue, TheWelcome.vue, WelcomeItem.vue та папку значків. Потім ми видалимо посилання з App.vue і очистимо шаблон.

Ви побачите папку __tests__ у папці компонентів, яка додається під час встановлення Vitest. Поки що можете ігнорувати це. В іншому випадку папка компонентів має бути порожньою.

Файл App.vue має виглядати так:

```
<script setup lang="ts">
</script>
<template>
</template>
<style scoped>
… (truncated, unchanged)
</style>
```

Зміни призведуть до порожньої сторінки, оскільки ми видалили всі елементи за замовчуванням! Тепер ми можемо почати створювати нашу власну програму з нуля.

Створення програми

У цьому розділі ми додамо кілька компонентів і створимо додаток Todo відповідно до вимог, перелічених на початку цього розділу (див. Технічні вимоги). Ми будемо додавати функції крок за кроком.

Почнемо просто з компонента AppHeader. Створіть файл ***AppHeader.vue*** (пам’ятайте: Vue.js рекомендує ім’я файлу, яке складається принаймні з двох слів із верблюжого регістру) у папці компонентів. Це буде лише статичний компонент із шаблоном і блоком CSS:

```
<template>
<header>
  <h1><span class="icon" aria-hidden="true">✔</span> To do</h1>
  <p>Building Real-world Web Applications with Vue.js 3</p>
</header>
</template>
```

```
<style scoped>
header {
  border-bottom: #333 1px solid;
  background-color: #fff;
}
header::after {
  content: "";
  display: block;
  height: 1px;
  box-shadow: 0px 0px 10px 0px rgba(0, 0, 0, 0.75);
}
h1 {
  font-size: 2rem;
}
h1 .icon {
  font-size: 1rem;
  vertical-align: middle;
}
</style>
```

Компонент Vue зазвичай складається з блоків шаблону, сценарію та стилю. Не всі є обов'язковими (ми не додаємо жодних сценаріїв до цього компонента), але серед цих трьох ми можемо визначити кожен аспект нашого компонента. У цій книзі ми побачимо численні приклади цього шаблону на практиці.

Шаблон — це лише представлення заголовка, і ми використаємо блок стилю з обмеженою областю, щоб застосувати правила CSS до розмітки. Зверніть увагу на атрибут scoped, який гарантує, що наш CSS не впливає на інші компоненти програми.

Використовуючи CSS з областю видимості, ми можемо писати чисті, читабельні правила. Для однофайлових компонентів цей підхід має бути типовим.

Давайте продовжимо створювати наш додаток, створивши компонент списку.

Створення компонента ListItem

Ми створимо компонент ListItem.vue у тій же папці. Це буде представлення окремого елемента в списку, і воно виглядає так:

```
<script lang="ts" setup>
defineProps<{
  isChecked?: boolean | false
}>()
</script>
<template>
  <label :class="{ 'checked': isChecked }">
    <input type="checkbox" :checked="isChecked" />
    <slot></slot>
  </label>
</template>
<style scoped>
label {
  cursor: pointer;
}
.checked {
  text-decoration: line-through;
}
</style>
```

Ми визначаємо властивості, які будуть передані компоненту. Props – це властивості, якими можна керувати ззовні компонента. Зазвичай це значення, які обробляються компонентом і визначають унікальні характеристики компонента в цьому стані. У рідкісних випадках ви також можете передати функцію.

Використовуючи метод defineProps, ми використовуємо API Vue.js для належного оголошення наших пропів.

Другий біт — це спосіб, яким компонент має відтворювати HTML у віртуальному DOM. Vue.js використовує синтаксис на основі HTML. Ви можете прочитати більше про це тут: https://vuejs.org/guide/essentials/template-syntax.html#template-syntax .

Ми розмічаємо HTML-тег <label> динамічним іменем класу: воно відображатиметься як class="checked", коли властивість isChecked оцінюється як true. До мітки ми додамо прапорець із динамічним перевіреним атрибутом: він також пов’язаний із властивістю isChecked. Тег <slot></slot> є специфічним для Vue.js і дозволяє нам розмістити будь-який вміст у цьому місці з батьківського компонента.

Нарешті, ми визначаємо правила CSS для цього компонента, подібно до того, що ми зробили з AppHeader.vue.

Створення списку

З наявним у нас компонентом ListItem ми можемо почати генерувати список. Для цього ми створимо новий компонент, який зберігатиме інформацію про список і використовуватиме його для відтворення всіх окремих елементів у списку, а також для надання інтерактивних функцій:

1. Давайте створимо простий файл під назвою TodoList.vue з таким вмістом:

```
<script lang="ts" setup>
import ListItem from './ListItem.vue'
</script>
<template>
<ul>
  <ListItem :is-checked="false">This is the slotted content</
ListItem>
</ul>
</template>
```

2. Перш ніж продовжити, ми хочемо мати можливість відображати нашу програму, поки ми над нею працюємо. Отже, у файлі App.vue дотримуйтеся аналогічного підходу до імпорту AppHeader.vue і TodoList. файли vue і додавання компонентів до шаблону:

```
<script setup lang="ts">
import AppHeader from './components/AppHeader.vue';
import TodoList from './components/TodoList.vue';
</script>
<template>
  <AppHeader />
  <TodoList />
</template>
<style scoped>
… (truncated)
</style>
```

Тепер, коли ми бачимо, що ми робимо, настав час запустити сервер розробки. Якщо ви використовуєте Visual Studio Code (VSCode), у середовищі IDE є вбудований термінал:

• користувачі macOS: ^ + `

• Користувачі Windows: Ctrl + `

• Користувачі Linux: Ctrl + Shift + `

Якщо ви запустите команду **npm run dev**, вона запустить сервер розробки та надасть вам локальну URL-адресу попереднього перегляду.

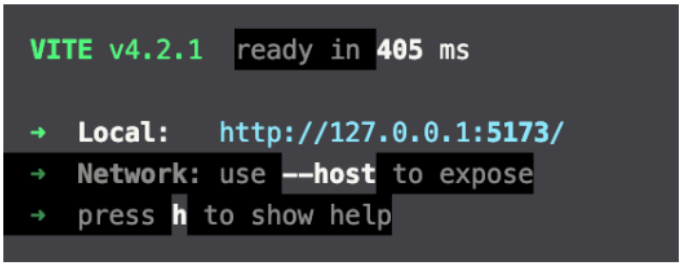

Рисунок 2.2 – Вихід команди npm run dev

Оскільки зараз у нас немає функціональної програми, нам потрібно попрацювати над її основною функцією: списком. Ви можете залишити сервер розробки запущеним, оскільки він автоматично оновлюватиметься (це називається гарячим перезавантаженням) із нещодавно написаним кодом.

Складання списку

Фактична функція міститься в компоненті TodoList.vue, який ми зараз створимо. Ми почнемо з малого й поступово додамо складніші функції. Почнемо зі статичного списку, який має кілька статусів списку.

Давайте спочатку розглянемо блок сценарію. Окрім імпортування компонента ListItem, ми визначаємо тип для Item, який складається з властивості title у вигляді рядка та додаткової властивості, перевіреної як Boolean. TypeScript дозволяє нам визначити псевдонім Type, до якого наша IDE може підключатися під час взаємодії з Type.

У цьому прикладі під час доступу до властивостей елемента в шаблоні ListItem IDE вже розпізнає заголовок і додаткові перевірені властивості:

```
<script lang='ts' setup>
import ListItem from './ListItem.vue'
type Item = {
  title: string,
  checked?: boolean
}
```

```
const listItems: Item[] = [
  { title: 'Make a todo list app', checked: true },
  { title: 'Predict the weather', checked: false },
  { title: 'Play some tunes', checked: false },
  { title: 'Let\'s get cooking', checked: false },
  { title: 'Pump some iron', checked: false },
  { title: 'Track my expenses', checked: false },
  { title: 'Organize a game night', checked: false },
  { title: 'Learn a new language', checked: false },
  { title: 'Publish my work' }
]
</script>
```

Під час створення масиву ListItems ми призначаємо Type масиву цього типу за допомогою символів []. Ми одразу заповнюємо масив ListItems списком елементів. Це означає, що TypeScript також може виводити типи, але краще явно встановлювати типи, де це можливо.

У шаблоні ми створюємо невпорядкований елемент списку та використовуємо ***директиву v-for*** для повторення елементів у масиві:

```
<template>
  <ul>
    <li
      :key='key'
      v-for='(item, key) in listItems'
    >
<ListItem :is-checked='item.checked'>{{ item.title }}</ListItem>
    </li>
  </ul>
</template>
```

Директива v-for використовується для циклічного перегляду колекцій і повторення шаблону, який позначає колекцію. Для кожного елемента поточне значення присвоюється першому аргументу (елементу) і додатково надає індекс колекції як другий аргумент (ключ).

Директива v-for повторює елемент <li> із включеним компонентом <ListItem />. Для кожного елемента ми заповнюємо компонент <ListItem /> властивостями is-checked і title для цього елемента.

Атрибут key допомагає Vue.js відстежувати зміни, які вносяться, щоб він міг ефективніше оновлювати віртуальну DOM.

Нарешті, ми додали блок стилів із обмеженою областю, щоб стилізувати елементи для браузера. Тут нічого не відбувається:

```
<style scoped>
ul {
  list-style: none;
}
li {
  margin: 0.4rem 0;
}
</style>
```

Тепер у нас є неінтерактивна програма зі списком завдань, і ми вже виконали перші дві вимоги. Давайте розглянемо, як ми можемо додати трохи інтерактивності.

Пояснення реактивності

Якщо ви відкрили програму та клацнули елемент, ви можете встановити прапорець, але після оновлення сторінки нічого не відбувається. Крім того, якщо ви уважно подивилися на CSS компонента <ListItem />, ви могли помітити, що закреслений стиль слід застосовувати до позначеного елемента. Це стосується лише першого пункту.

Перемикання прапорця – це, по суті, поведінка рідного браузера і нічого не означає в контексті стану списку Todo!

Нам потрібно пов’язати зміни в інтерфейсі користувача зі станом програми. Щоб почати, нам потрібно імпортувати деякі утиліти з пакета Vue.js. Додайте ці два рядки у верхній частині блоку <script>:

```
import { ref } from 'vue'
import type { Ref } from 'vue'
```

Функція ref використовується для додавання реактивності та відстеження оновлень певних частин коду. Значення ref автоматично виводиться TypeScript, але для складних типів ми можемо вказати тип.

Примітка

Vue.js також пропонує утиліту reactive для позначення реактивності. Між ними є невеликі відмінності, де ref можна використовувати для відстеження примітивів і об’єктів, а reactive можна ініціалізувати лише

об’єктом. Загалом, ви можете обрати узгодженість коду, вибравши ref замість reactive. Єдиним недоліком є те, що вам потрібно отримати доступ до значення реактивного елемента за допомогою властивості .value у блоці сценарію. Під час використання змінної в блоці шаблону Vue.js автоматично розгортає її. Таким чином, це невелика поступка за можливість постійного використання ref.

Тепер, коли ми імпортували утиліту, ми можемо позначити ListItems для відстеження, загорнувши вміст у функцію ref:

```
const listItems: Ref<Item[]> = ref([
  { title: 'Make a todo list app', checked: true },
  { title: 'Predict the weather', checked: false },
  { title: 'Play some tunes', checked: false },
  { title: 'Let\'s get cooking', checked: false },
  { title: 'Pump some iron', checked: false },
  { title: 'Track my expenses', checked: false },
  { title: 'Organise a game night', checked: false },
  { title: 'Learn a new language', checked: false },
  { title: 'Publish my work' }
])
```

Зауважте, що Ref з великої літери використовується для введення значення, а ref з нижнього регістру використовується як оболонка для масиву елементів. Якщо тепер ми хочемо отримати доступ до значень у блоці сценарію, нам потрібно отримати до них доступ за допомогою listItems.value.

Тепер, коли елементи списку є реактивними, віртуальний DOM автоматично реагуватиме на зміни змінних. Ми можемо додати метод, який змінює елемент так, щоб він відображався в інтерфейсі користувача.

Давайте додамо таку функцію до блоку сценарію:

```
const updateItem = (item: Item): void => {
  const updatedItem = findItemInList(item)
  toggleItemChecked(updatedItem)
}
const findItemInList = (item: Item): Item | undefined => {
  return listItems.value.find(
    (itemInList: Item) => itemInList.title === item.title
)
}
const toggleItemChecked = (item: Item): void => {
```

```
  item.checked = !item.checked
}
```

Прийнявши філософію «Чистого коду» Роберта С. Мартіна, я розділив інструкцію на окремі функції з їхнім чітким наміром. Під час виклику updateItem із item як аргументом він намагатиметься знайти його в itemList і перемкне позначену властивість об'єкта.

Ми бачимо, що TypeScript направляє нас до трохи кращого рішення: оскільки findItemInList може повертати невизначене значення, а toggleItemChecked очікує параметр, аргумент виклику функції toggleItemChecked отримує хвилясту лінію (squiggly line).

Рисунок 2.3 – TypeScript натякає на можливу проблему в нашому коді

Ми можемо виправити це, додавши оператор навколо виклику функції toggleItemChecked:

```
const updateItem = (item: Item): void => {
  const updatedItem = findItemInList(item)
  if (updatedItem) {
    toggleItemChecked(updatedItem)
  }
}
```

Після завершення змін блоку сценарію ми можемо приєднати інтерактивність до інтерфейсу користувача в блоці шаблону. Ми хочемо, щоб відвідувач міг натиснути на ListItem, щоб позначити його як завершений.

Vue.js має для цього вбудовану ***директиву: v-on***. Це діє як обробник подій, а також підтримує кілька модифікаторів. Для отримання додаткової інформації див. https://vuejs.org/api/built-indirectives.html#v-on .

Ми можемо додати його до шаблону так:

```
<ListItem :is-checked='item.checked' v-on:click.
prevent="updateItem(item)">{{ item.title }}</ListItem>
```

Ми також додали модифікатор *.prevent*, щоб запобігти типовій поведінці механізму прапорця. Це весь код, який потрібен для виклику методу!

Існує навіть скорочення для *v-on:click* за допомогою *@click*. Ви побачите приклади обох директив у ресурсах, тому добре розуміти, що вони однакові.

Під капотом Vue.js використовує функцію ref для реєстрації серії спостерігачів за значеннями. Механізм шаблонів використовується для створення віртуальної DOM (представлення дерева вузлів елементів, які складають компонент). Після зміни реактивного значення також змінюється віртуальний вузол DOM, де це значення використовується.

Vue.js порівнює зміни в DOM і оновлює лише необхідні елементи в реальному DOM, щоб відобразити стан. Можливість дуже точного деталізованого оновлення DOM робить Vue.js 3 високопродуктивним фреймворком, оскільки йому не потрібно проходити цілі гілки віртуальних вузлів DOM!

Давайте використаємо список, який у нас є, як вхідні дані для нашого наступного кроку, де ми розглянемо сортування.

Сортування списку

Тепер ми цілком можемо показувати змінні в шаблоні. Однак бувають випадки, коли вам потрібні більш складні вирази, наприклад, у нашому прикладі вимога сортування списку. Для змінних, які не мають побічних ефектів і містять реактивні дані, ви можете використовувати computed функцію Vue.js.

Як правило, ви використовуєте computed для фільтрації даних, форматування виразів, відображення обчислень або логічних умов. Давайте застосуємо його, щоб відсортувати список із завершеними елементами внизу.

Спочатку ми імпортуємо обчислені до компонента TodoList. Ми можемо додати його до імпорту, де ми також імпортуємо функцію ref:

```
import { ref, computed } from 'vue'
```

Обчислена функція дуже схожа на ref у тому сенсі, що вона має ту саму реакцію в оновленні DOM, коли значення змінюється, і ви навіть можете отримати доступ до значення за допомогою властивості .value у блоці

сценарію! Основна відмінність полягає в тому, що обчислене значення кешується й оновлюється лише тоді, коли змінюється один із вхідних даних.

Для сортування списку ми можемо використовувати listItems як вхідні дані та застосувати до масиву просту функцію сортування JavaScript. Ми можемо просто додати цей рядок, щоб визначити обчислене значення:

```
const sortedList = computed(() =>
[...listItems.value].sort((a, b) => (a.checked ? 1 : 0) -
(b.checked ? 1 : 0))
)
```

Як бачите, computed — це функція, яка викликається при зміні реактивного значення. У цьому випадку listItems.value. Ми просто застосуємо функцію сортування до колекції.

Тепер у шаблоні ми можемо замінити listItems на змінну sortedList, і ви побачите, що позначені елементи будуть розміщені під непозначеними.

Збереження змін до списку

Зараз у нас є остання вимога, а саме збереження стану списку під час перезавантаження та повторного перегляду програми. Наразі ми зробимо це максимально простим і використаємо API localStorage веб-браузера для зберігання та отримання стану списку.

Спочатку ми додамо функції, які ми можемо використовувати для запису в localStorage та отримання з localStorage:

```
const setToStorage = (items: Item[]): void => {
  localStorage.setItem('list-items', JSON.stringify(items))
}
const getFromStorage = (): Item[] | [] => {
  const stored = localStorage.getItem('list-items')
  if (stored) {
    return JSON.parse(stored)
  }
  return []
}
```

Ці дві функції взаємодіють зі здатністю браузерів зберігати рядок даних, тому нам потрібно створити рядок і проаналізувати цей об’єкт. Ми зберігаємо дані в ключі елементів списку.

Тепер нам потрібно переконатися, що ми намагаємося отримати дані, коли компонент завантажується. Для нього є функція, яка називається *onMounted* і є частиною ядра Vue.js, тому ми можемо імпортувати його подібним чином до функцій ref і computed.

Функцію *onMounted* ми називаємо хуком життєвого циклу. Це функції, які викликаються в певні моменти життєвого циклу компонента. Основні події життєвого циклу запускаються, коли компонент монтується (або раніше), оновлюється (або раніше), демонтується (або раніше) і видає помилку. Більше інформації можна знайти тут: https://vuejs.org/api/composition-api-lifecycle.html#composition-api-lifecycle-hooks .

У нашому випадку ми хочемо, щоб список був отриманий у браузері під час рендерингу компонента (інакше він не мав би доступу до localStorage). Отже, ми імпортуємо функцію:

```
import { ref, onMounted, computed } from 'vue'
```

І нам потрібно створити реактивну змінну для зберігання елементів:

```
const storageItems: Ref<Item[]> = ref([])
```

Ми також створимо функцію (initListItems), яка запускатиметься один раз після монтування, і перемістимо туди ініціалізацію listItems. Ми також внесемо зміни в оголошення listItems, обернувши його перевіркою на існування storageItems. Якщо вони не існують, ми будемо використовувати listItems за замовчуванням і запишемо вміст у localStorage:

```
const initListItems = (): void => {
  if (storageItems.value?.length === 0) {
    const listItems = [
      { title: 'Make a todo list app', checked: true },
      { title: 'Predict the weather', checked: false },
      { title: 'Read some comics', checked: false },
      { title: 'Let\'s get cooking', checked: false },
      { title: 'Pump some iron', checked: false },
      { title: 'Track my expenses', checked: false },
      { title: 'Organise a game night', checked: false },
      { title: 'Learn a new language', checked: false },
      { title: 'Publish my work' }
  ]
    setToStorage(listItems)
    storageItems.value = listItems
```

```
  }
}
```

Тепер ми додаємо наступні функції для отримання будь-яких локально збережених елементів списку:

```
onMounted(() => {
  initListItems()
  storageItems.value = getFromStorage()
})
```

Щоб підтримувати синхронізацію змін, тепер ми можемо змінити функцію findItemInList, щоб шукати в колекції storageItems, а не в listItems, а також записати зміни в сховище після оновлення елемента. Ми змінимо функції updateItem і findItemInList наступним чином:

```
const updateItem = (item: Item): void => {
  const updatedItem = findItemInList(item)
  if (updatedItem) {
    toggleItemChecked(updatedItem)
    setToStorage(storageItems.value)
  }
}
const findItemInList = (item: Item): Item | undefined => {
  return storageItems.value.find(
    (itemInList: Item) => itemInList.title === item.title
  )
}
```

Тепер у шаблоні ми використовуємо обчислене значення, тому нам також слід оновити обчислену функцію, щоб побачити localStorage як вхідні дані для наших даних:

```
const sortedList = computed(() =>
  [...storageItems.value].sort((a, b) => (a.checked ? 1 : 0) -
  (b.checked ? 1 : 0))
)
```

Ми побачили, як ми можемо використовувати різні компоненти з особливим призначенням для створення простого реактивного додатка та як ми можемо організувати наш код з урахуванням зручності читання та обслуговування. Vue.js заохочує використовувати однофайлові компоненти для структурування коду.

Компоненти одного файла

Те, як ми організували програму, коли окремі компоненти мають одну функцію, називається філософією **Single File Components (SFC).**

Цей підхід призначений для покращення читабельності коду, підтримки та можливості повторного використання. За допомогою SFC ви можете створювати повторно використовувані та модульні компоненти, якими можна легко ділитися та повторно використовувати в різних проектах.

Справедливості заради, ми все-таки зрізали деякі кути з компонентом TodoList.vue, оскільки ми могли абстрагувати отримання та налаштування listItems іншим компонентом. Однак для цього прикладу він ілюструє можливості в прийнятний спосіб. Не існує строгих правил чи вказівок щодо того, як ви структуруєте свої компоненти.

Зауважте, що ви можете структурувати або реструктурувати вміст блоку сценарію так, як вам буде зрозуміло. У вас є свобода групувати пов’язані набори разом, що робить код дуже читабельним, який легко рефакторювати.