LARAVEL X VUE Conf Taiwan 2022

Vue.js 實戰工作坊
Vue 3.x 全家桶打造仿 Trello 的任務看板

Kuro Hsu

LARAVEL X VUE Conf Taiwan 2022

About Me - Kuro Hsu

- 👨‍💻 前端開發者，喜歡各種浮誇的玩具
- 📝 出過兩本書，其中一本正在改版中
- Vue.js Taiwan 社群掛名雜工
- 🏠 https://kuro.tw
- 📩 kurotanshi [at] gmail.com

先看成品


Hello Vue.js!
Vue.js 測試
內文區塊
Vue.js 測試2
內文區塊2
+ 點擊以新增任務
Hello Laravel!
Laravel 測試
內文區塊
Laravel 測試2
內文區塊2
+ 點擊以新增任務
+ 新增其他列表

技術架構

- Vite 3.1.8
- Vue.js 3.2
- Vue Router 4.1.5
- Pinia 2.0
- Vue Use
- Tailwindcss 3.2.14
- Vue.draggable.next
- unplugin-vue-components

建立專案

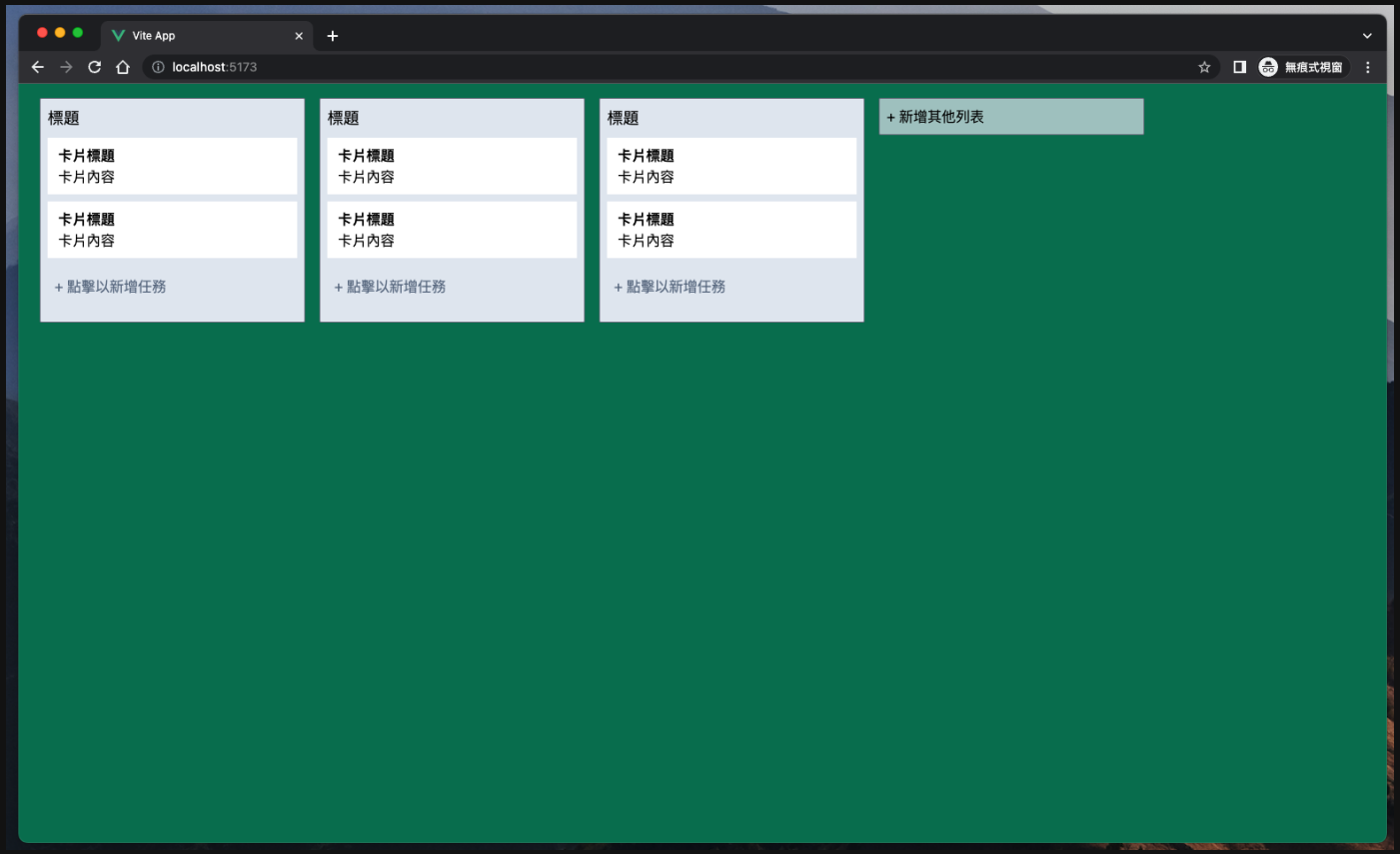
- 確定已安裝 Node.js v16 以上版本
- 打開終端機，輸入以下指令

```
# trello-app 可以改成任何你想要的專案名稱
npx degit kurotanshi/vue-workshop-trello trello-app
```

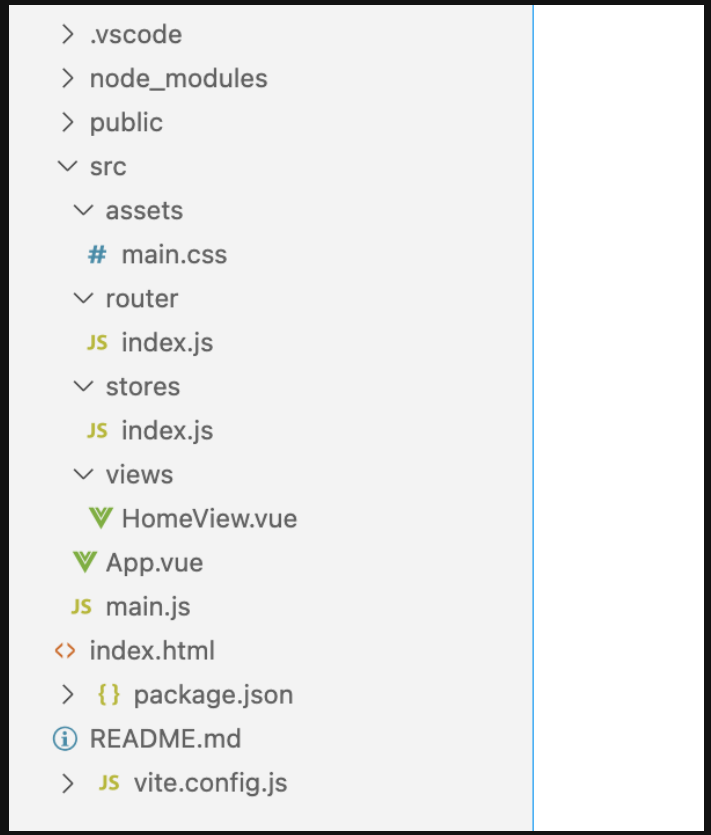
```
cd trello-app
npm install
npm run dev
```

此專案內相關的套件、及需要的設定檔都已經預先安裝好。

啟動後的畫面


Vite App
localhost:5173
無痕式視窗
標題
卡片標題
卡片內容
卡片標題
卡片內容
+ 點擊以新增任務
標題
卡片標題
卡片內容
卡片標題
卡片內容
+ 點擊以新增任務
標題
卡片標題
卡片內容
卡片標題
卡片內容
+ 點擊以新增任務
+ 新增其他列表

目錄結構

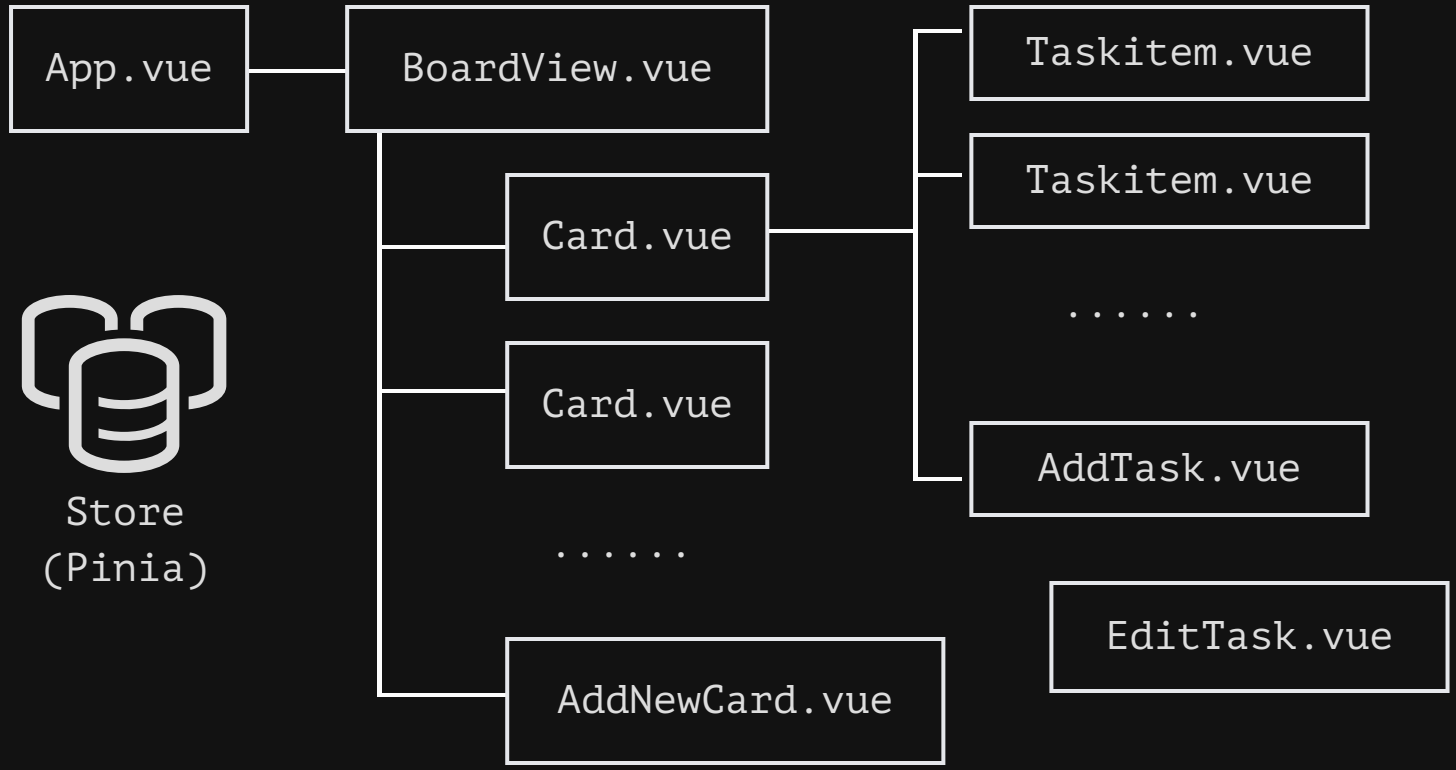


- `/src/` : Vue 主要的原始檔目錄
- `/views/` : 頁面元件目錄
- `/components/` : 子元件目錄
- `/store` : Pinia Store
- `/router` : Vue Router 設定
- `vite.config.js` : Vite 設定檔

今日目標

- 將 App.vue 內的結構根據不同功能拆分元件
- Pinia 將元件共用狀態抽離
- Tailwind CSS 快速切換 UI 狀態
- Vue Use 提供的 `focus` 與 `onClickOutside` 處理元素焦點
- Vue.draggable.next 建立拖拉功能
- Vue Router 建立路由，可根據網址切換頁面
- 將列表儲存至 localStorage，重新整理後仍可保留資料

完整的專案結構


App.vue
BoardView.vue
Taskitem.vue
Taskitem.vue
Card.vue
.....
Card.vue
AddTask.vue
Store
(Pinia)
.....
EditTask.vue
AddNewCard.vue

建立基礎路由

- 首先將 `App.vue` 的內容搬移到 `src/views/BoardView.vue` 檔案中
- 在 `App.vue` 的 `<template>` 區塊加入 `<router-view />`，讓路由可以切換頁面
- 接著修改 `src/router/index.js`，將 `BoardView` 設定為預設頁面

```
routes: [
  {
    path: "/",
    component: () => import("/src/views/BoardView.vue"),
  },
],
```

拆分元件

拆分元件 - Card.vue

- 在 `src/` 建立 `components` 目錄，並新增 `Card.vue` 檔案
- 接著開啟 `src/views/BoardView.vue`，將 Card 區塊搬移至 `Card.vue` 的 `<template>` 區塊內
- 移除原本的 `v-for="i in 3" :key="i" @click="toggle = !toggle"`
- `BoardView.vue` 檔案原本的區塊改為 `<Card />`

拆分元件 - AddNewTask.vue

- 在 `src/components` 目錄新增 `AddNewTask.vue` 檔案
- 將 `Card.vue` 中的 `add new task` 區塊搬移至 `AddNewTask.vue` 的 `<template>` 區塊內
- `Card.vue` 檔案原本的 `add new task` 區塊改為 `<AddNewTask />`

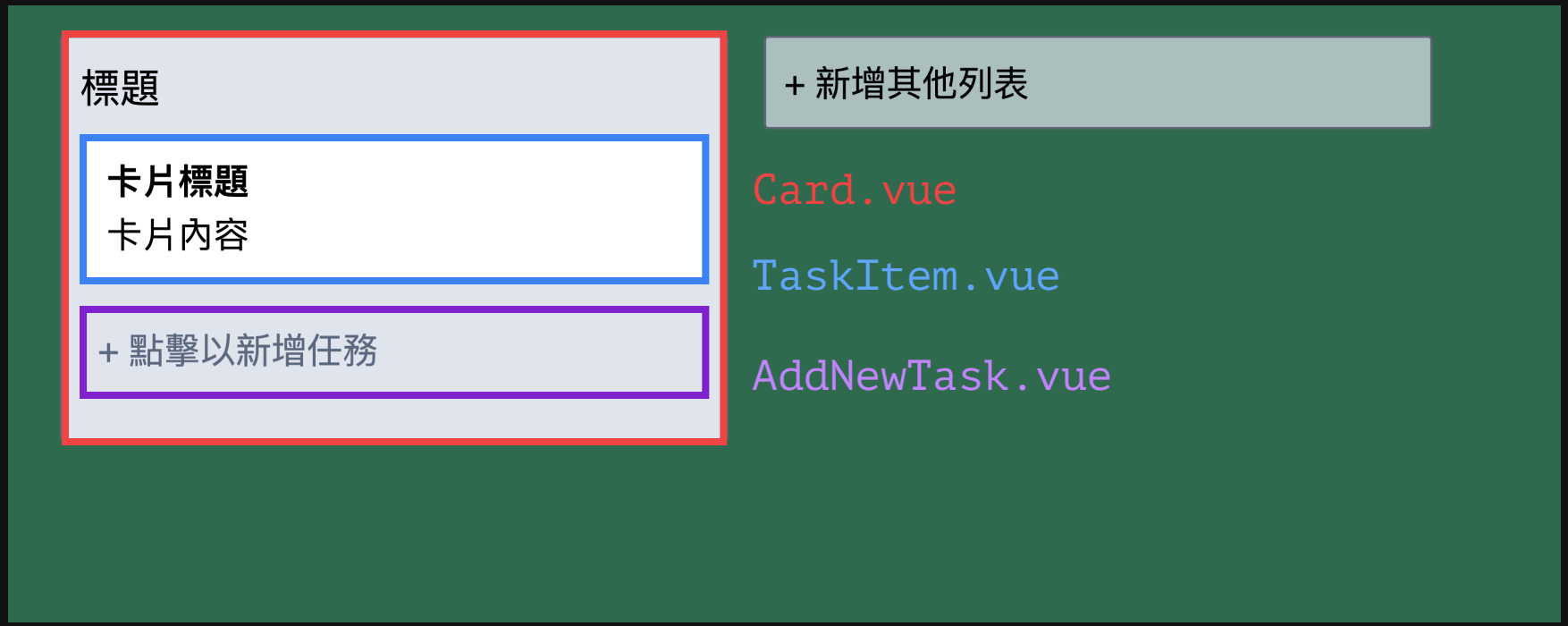
拆分元件 - TaskItem.vue

- 在 `src/components` 目錄新增 `TaskItem.vue` 檔案
- 仿造前面兩個元件的做法，將卡片的部分移到 `TaskItem.vue` 的 `<template>` 區塊內
- `Card.vue` 檔案原本的卡片區塊改為 `<TaskItem />`

Card.vue

```vue
<script setup></script>

<template>
  <div
    class="border rounded-sm bg-slate-200 border-gray-500 mx-2 min-w-[300px] p-2 block"
  >
    <!-- column -->
    <div class="text-ellipsis text-lg w-4/5 block overflow-hidden">標題</div>
    <!-- <textarea class="border-none h-8 w-full p-1 resize-none overflow-hidden block"></textarea> -->

    <!-- tasks -->
    <TaskItem />
    <!-- tasks -->

    <!-- add new task -->
    <AddNewTask />
    <!-- add new task -->
  </div>
</template>
```

Card.vue


標題
卡片標題
卡片內容
+ 點擊以新增任務
+ 新增其他列表
Card.vue
TaskItem.vue
AddNewTask.vue

為什麼元件不需要 `import` ？

- `unplugin-vue-components` 會自動將目前元件內所需要的子元件，根據名稱自動引入
- 要小心重複命名問題

```
// vite.config.js
import { defineConfig } from "vite";
import vue from "@vitejs/plugin-vue";
import Components from "unplugin-vue-components/vite";

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    vue(),
    Components({
      /* options */
    }),
  ],
  // 後略
});
```

建立共用狀態

建立共用狀態

- 開啟 `stores/index.js` ，修改 `useStore` 的內容，將 `lists` 回傳

```
// stores/index.js

import { ref } from "vue";
import { defineStore } from "pinia";

const defaultList = [
  // 略
];

export const useStore = defineStore("store", () => {
  const lists = ref(defaultList);

  return {
    lists,
  };
});
```

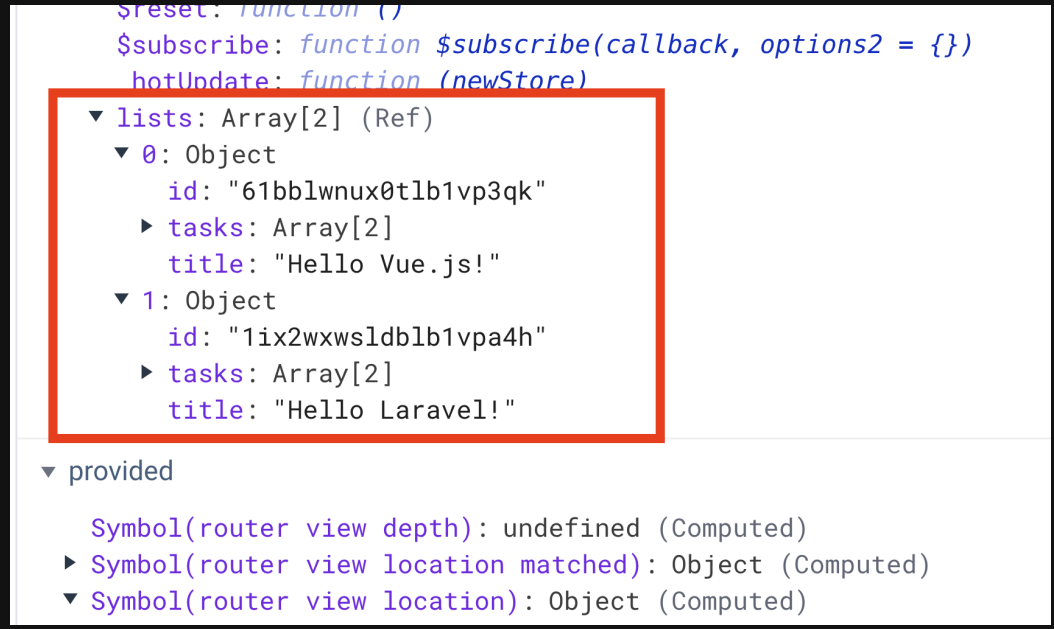
將 `lists` 傳遞給元件

- 回到 `BoardView.vue` 修改 `<script>` 區塊

```
<script setup>
import { ref, computed } from "vue";
import { useStore } from "/src/stores";

const store = useStore();
const list = computed(() => store.lists);

const toggle = ref(false);
</script>
```

將 `lists` 傳遞給元件

- 如圖，可開啟 vue devtool 確定 `BoardView.vue` 能正確取得 `lists` 陣列的內容



改寫 `BoardView.vue`

- 改寫 `<Card>` 加上 `v-for` 指令
- 並將 `card` 透過 props 方式傳遞給 `Card.vue`

```
<Card v-for="card in list" :key="card.id" v-bind="card" />
```

回到 `Card.vue`

- 加上 `defineProps` 來接收 `BoardView.vue` 傳遞的 props

```
<script setup>
import { ref, computed } from "vue";

const props = defineProps({
  id: String,
  title: String,
  tasks: Array,
});
</script>
```

- 加上標題的顯示：

```
<div class="text-ellipsis text-lg w-4/5 block overflow-hidden">
  {{ title }}
</div>
```

`Card.vue`


Hello Vue.js!
卡片標題
卡片內容
+ 點擊以新增任務
Hello Laravel!
卡片標題
卡片內容
+ 點擊以新增任務
+ 新增其他列表

`Card.vue`

- 加上標題的編輯狀態切換

```
<script setup>
import { ref, computed } from "vue";

const props = defineProps({
  id: String,
  title: String,
  tasks: Array,
});

// 注意，不可直接修改 props 的值
const title = ref(props.title);
const isTitleEditing = ref(false);
</script>
```

`Card.vue`

- 加上標題的編輯狀態切換

```
<!-- column -->
<div
  v-if="!isTitleEditing"
  @click="isTitleEditing = true"
  class="text-ellipsis text-lg w-4/5 block overflow-hidden"
>
  {{ title }}
</div>
<textarea
  v-else
  v-model="title"
  @keydown.enter="isTitleEditing = false"
  class="border-none h-8 w-full p-1 resize-none overflow-hidden block"
></textarea>
```

`Card.vue`

- 點擊時自動進入輸入框焦點
  - 利用 VueUse 提供的 `useFocus`
- 點擊輸入框以外的範圍自動關閉編輯狀態
  - 利用 VueUse 提供的 `vOnClickOutside`

```
import { useFocus } from "@vueuse/core";
import { vOnClickOutside } from "@vueuse/components";
```

`Card.vue`

- 點擊時自動進入輸入框焦點

```
import { useFocus } from "@vueuse/core";

const target = ref();
useFocus(target, { initialValue: true });
```

```
<textarea
  v-else
  ref="target"
  v-model="title"
  @keydown.enter="isTitleEditing = false"
  class="border-none h-8 w-full p-1 resize-none overflow-hidden block"
></textarea>
```

`Card.vue`

- 點擊輸入框以外的範圍自動關閉編輯狀態
- 在 `script setup` 的語法中， `v` 開頭的駝峰名稱會被自動轉換為 `v-` 的指令

```
import { vOnClickOutside } from "@vueuse/components";
```

```
<textarea
  v-else
  v-model="title"
  ref="target"
  @keydown.enter="isTitleEditing = false"
  v-on-click-outside="() => (isTitleEditing = false)"
  class="border-none h-8 w-full p-1 resize-none overflow-hidden block"
></textarea>
```

將修改後的標題回存到 `store`

- 在 `store` 新增 `updateListTitle` 方法，用來更新 `list` 的標題

```
export const useStore = defineStore("store", () => {
  const lists = ref(defaultList);

  const updateListTitle = (cardId = "", title = "") => {
    const card = lists.value.find((list) => list.id === cardId);
    card.title = title;
  };

  return {
    lists,
    updateListTitle,
  };
});
```

`Card.vue`

- 在 `Card.vue` 中，利用 `useStore` 來取得 `store` 的方法

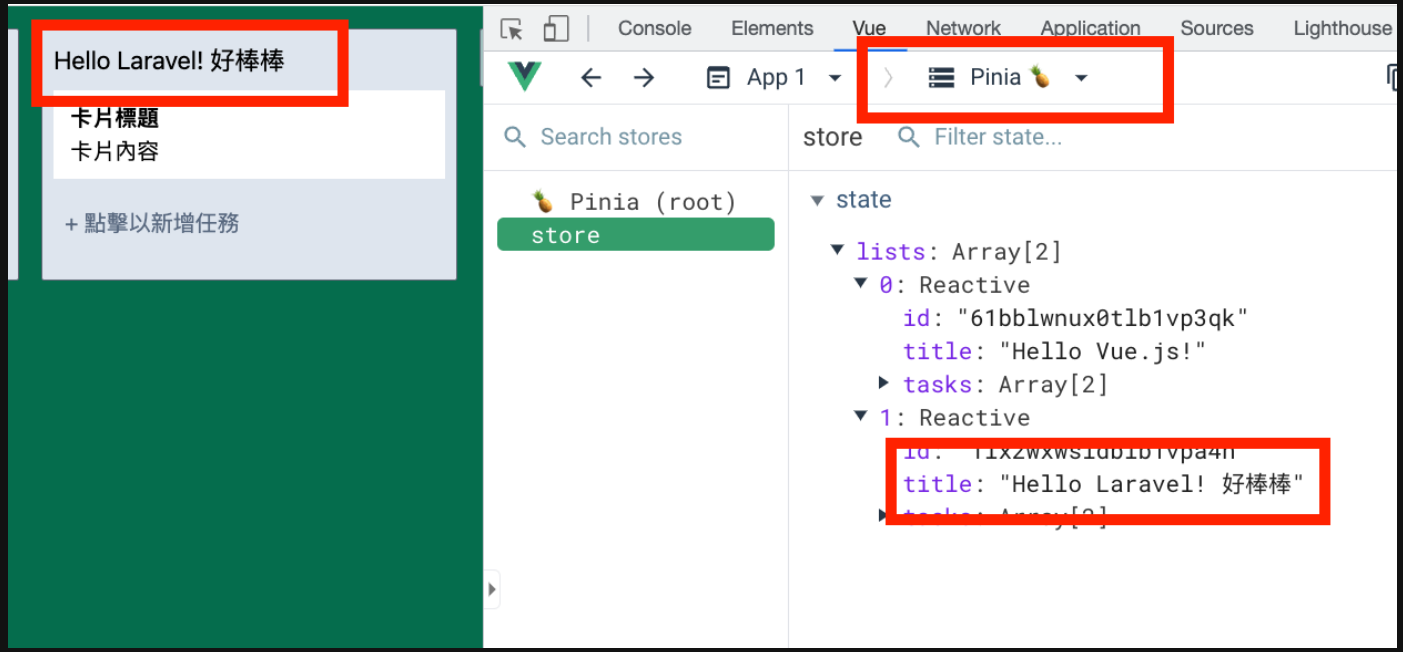
```
const store = useStore();
const { updateListTitle } = store;
```

- 加上 `watch` 當 `isTitleEditing` 狀態改變時，回存標題到 `store`

```
watch(isTitleEditing, (v) => {
  updateListTitle(props.id, title.value);
});
```

確認 Pinia 有寫入標題

- 這樣標題的更新就完成了



Break time

更新任務列表內容

- 改寫 `Card.vue` 的 `<TaskItem />` 區塊，並將 `task` 透過 props 傳入
- `v-bind` 會將 `task` 的所有屬性解構後傳入 `<TaskItem />` 內

```
<TaskItem v-for="task in tasks" :key="task.id" v-bind="task" />
```

更新任務列表內容

- 在 `TaskItem.vue` 中，加上 `props` 的型別定義
- 因為不需要在此編輯，所以無需加上 `ref`

```
<script setup>
const props = defineProps({
  id: String,
  title: String,
  content: String,
});
</script>

<template>
  <div class="bg-white my-2 w-full py-2 px-3 block overflow-hidden select-none">
    <div class="font-bold block">{{ title }}</div>
    <div class="text-ellipsis overflow-hidden">{{ content }}</div>
  </div>
</template>
```

更新任務列表內容

- 這樣顯示的部分就完成了

新增任務

- 修改 `AddNewTask.vue` 檔案，仿造 `Card.vue` 的修改標題的方式調整：

```
<script setup>
import { ref } from "vue";
import { useFocus } from "@vueuse/core";
import { vOnClickOutside } from "@vueuse/components";

const target = ref();
useFocus(target, { initialValue: true });

const newTitle = ref("");
const isEditing = ref(false);
</script>
```

新增任務

```
<template>
  <div class="my-3">
    <div
      v-if="!isEditing"
      @click="isEditing = true"
      class="cursor-pointer bg-slate-200 p-2 text-slate-500 hover:bg-slate-300"
    >
      + 點擊以新增任務
    </div>
    <textarea
      v-else
      ref="target"
      v-on-click-outside="() => (isEditing = false)"
      v-model="newTitle"
      @keydown.enter="isEditing = false"
      class="h-10 w-full p-2 block resize-none"
      placeholder="為這張卡片輸入標題"
    ></textarea>
  </div>
</template>
```

新增任務

- 同樣在 `stores/index.js` 新增 `addTask` 方法:

```
// 建立亂數 id
const uid = () => Math.random().toString(36).substring(2) + Date.now().toString(36);

// 新增任務
const addTask = (cardId = "", title = "") => {
  if (!cardId || !title) return;

  const card = lists.value.find((list) => list.id === cardId);
  card.tasks.push({
    id: uid(),
    title,
    content: "",
  });
};
```

新增任務

- 回到 `Card.vue` 將卡片 id 傳入 `<AddNewTask>` 元件：

```
<AddNewTask :id="props.id" />
```

新增任務

- 並在 `AddNewTask.vue` 中，加入 `Props` 取得傳入的 `id`
- 利用 `useStore` 取得 `store` 的 `addTask` 方法

```
import { useStore } from "/src/stores";

const props = defineProps({
  id: String,
});

const store = useStore();
const { addTask } = store;

// 儲存任務後清空輸入框
const addTaskToCard = () => {
  addTask(props.id, newTitle.value);
  newTitle.value = "";
  isEditing.value = false;
};
```

新增任務

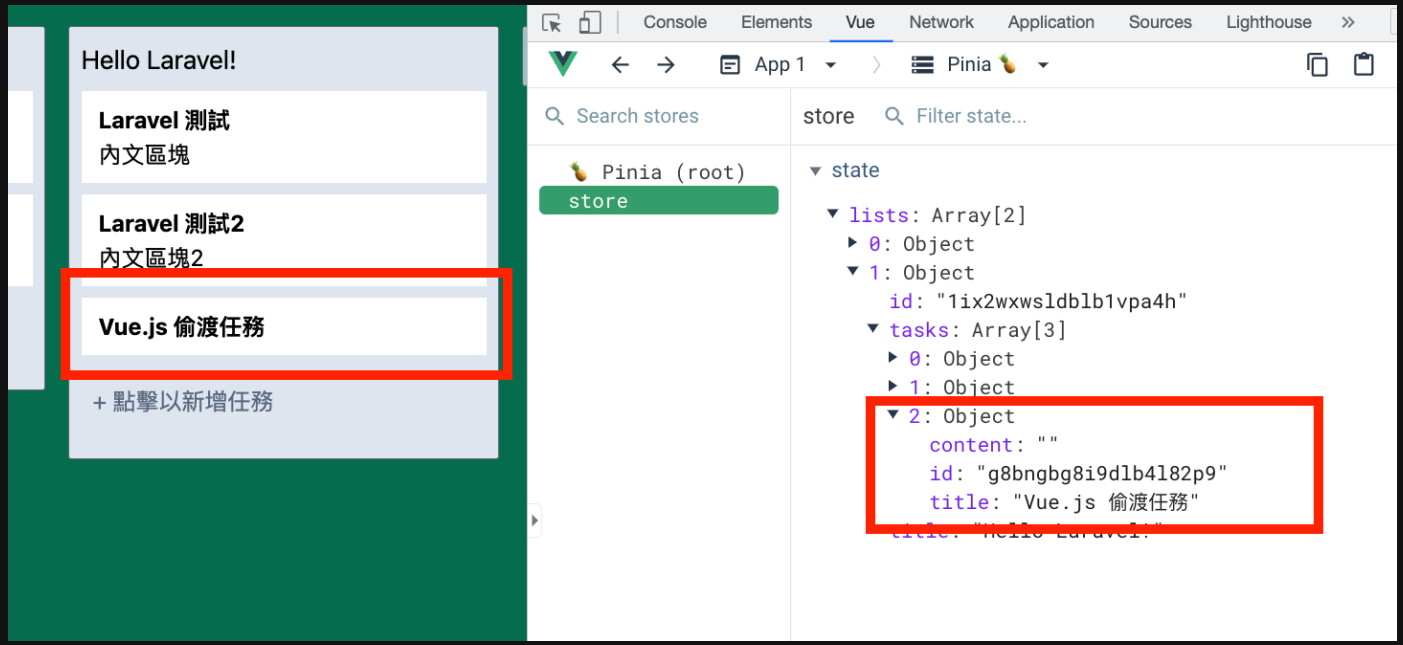
- 當按下 Enter 時，呼叫 `addTaskToCard` 方法

```
<textarea
  v-else
  ref="target"
  v-on-click-outside="() => (isEditing = false)"
  v-model="newTitle"
  @keydown.enter="addTaskToCard"
  class="h-10 w-full p-2 block resize-none"
  placeholder="為這張卡片輸入標題"
></textarea>
```

- 這裡不用 `watch` 的原因是反覆切換的時候會不斷新增

新增任務

- 這樣就完成了新增任務的功能

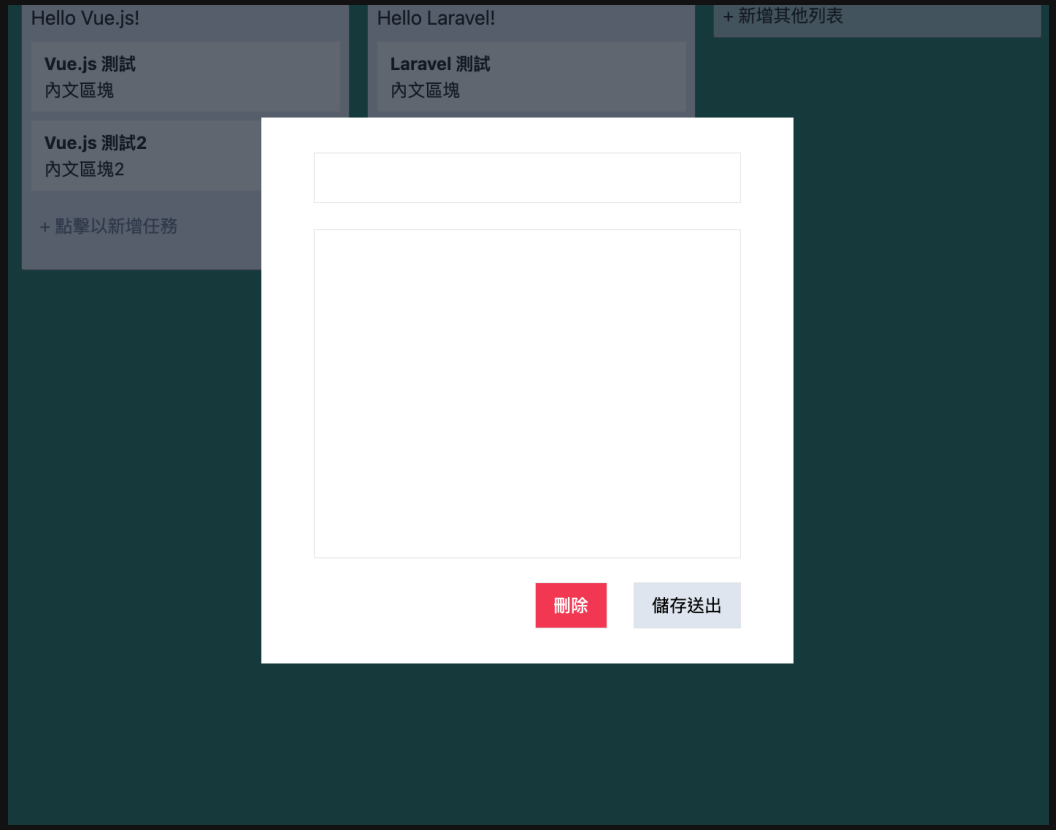


使用燈箱編輯卡片任務

編輯任務內容

- 在 `src/components` 目錄新增 `EditBox.vue` 檔案
- 回到 `views/BoardView.vue`，將 lightbox 部分的程式碼抽取出來，放到 `EditBox.vue` 中
- 移除 `v-if` 判斷
- 將原本的 lightbox 改為 `<EditBox>` 元件

編輯任務內容


Hello Vue.js!
Vue.js 測試
內文區塊
Vue.js 測試2
內文區塊2
+ 點擊以新增任務
Hello Laravel!
Laravel 測試
內文區塊
+ 新增其他列表
刪除
儲存送出

編輯任務內容

- 修改 `stores/index.js` ，建立 `currentEditTask` 狀態，用來判斷目前編輯的任務

```
const currentEditTask = ref({});

// 開啟編輯燈箱
const openEditTask = (cardId, taskId) => {
  const card = lists.value.find((list) => list.id === cardId);
  const task = card.tasks.find((task) => task.id === taskId);

  // 傳入卡片 id, 及任務資訊
  currentEditTask.value = {
    cardId,
    ...task,
  };
};

// 清空 currentEditTask 代表關閉燈箱
const closeEditTask = () => {
  currentEditTask.value = {};
};
```

編輯任務內容

- `views/BoardView.vue` 修改

```
// 從 store 取得 currentEditTask
const currentEditTask = computed(() => store.currentEditTask);
```

```
<!-- 判斷 currentEditTask.id 是否存在來決定燈箱顯示與否 -->
<EditBox v-if="currentEditTask?.id" />
```

編輯任務內容

- 調整 `Card.vue`

```
const store = useStore();
// 從 store 取得 openEditTask 方法
const { updateListTitle, openEditTask } = store;
```

```
<!-- 點擊任務區塊開啟燈箱 -->
<TaskItem
  v-for="task in tasks"
  :key="task.id"
  @click="openEditTask(props.id, task.id)"
  v-bind="task"
/>
```

編輯任務內容

- 調整 `EditBox.vue`

```
<script setup>
import { ref, computed } from "vue";
import { useStore } from "/src/stores";

// 從 store 取得 currentEditTask
const store = useStore();
const currentEditTask = computed(() => store.currentEditTask);

// 避免直接對 currentEditTask 修改，用 ref 包裝
const title = ref(currentEditTask.value.title);
const content = ref(currentEditTask.value.content);
</script>
```

編輯任務內容

- 調整 `EditBox.vue` ，在對應的欄位上加入 `v-model`

```
<div>
  <input
    ref="target"
    type="text"
    class="border text-xl mb-6 w-full p-2"
    v-model="title"
  />
</div>

<textarea
  class="border h-[300px] w-full p-3 overflow-x-hidden overflow-y-auto resize-none"
  v-model="content"
></textarea>
```

編輯任務內容

編輯任務內容

- 修改 `stores/index.js` ，新增 `updateTask` 方法

```
// 更新 task 內容
const updateTask = (cardId, taskId, title = "", content = "") => {
  const card = lists.value.find((list) => list.id === cardId);
  const task = card.tasks.find((task) => task.id === taskId);
  task.title = title;
  task.content = content;
  closeEditTask();
};
```

編輯任務內容

- 修改 `EditBox.vue` ，加入儲存事件

```
const { updateTask } = store;
```

```
<!-- 將 cardId, taskId, 與更新後的 title, content 送出 -->
<button
  @click="updateTask(currentEditTask.cardId, currentEditTask.id, title, content)"
  class="border bg-slate-200 py-2 px-4 hover:bg-slate-400 hover:text-slate-100"
>
  儲存送出
</button>
```

編輯任務內容

- 想要點擊燈箱外面的區塊關閉燈箱，除了使用 `v-click-outside` 套件之外，也可以利用 `@click.self`

```
const { updateTask, closeEditTask } = store;
```

- 關閉燈箱時不會儲存，只有在按下儲存按鈕時才會儲存

```
<!-- 在燈箱最外層加入 @click.self -->
<div
  class="h-full bg-slate-800 bg-opacity-70 w-full top-0 left-0 z-100 fixed"
  @click.self="closeEditTask"
>
<!-- 中間略 -->
</div>
```

刪除指定的任務

- 修改 `stores/index.js` ，新增 `deleteTask` 方法

```
// 刪除任務
const deleteTask = (cardId, taskId) => {
  const card = lists.value.find((list) => list.id === cardId);
  card.tasks = card.tasks.filter((task) => task.id !== taskId);
  closeEditTask();
};
```

刪除指定的任務

- 修改 `EditBox.vue` ，加入刪除事件

```
const { updateTask, closeEditTask, deleteTask } = store;
```

```
<button
  @click="deleteTask(currentEditTask.cardId, currentEditTask.id)"
  class="border bg-rose-500 text-white mr-6 py-2 px-4 hover:bg-rose-700">
  刪除
</button>
```

- 這樣就完成編輯與刪除任務的功能了

Break time

新增卡片 (新增其他列表)

- 在 `src/components/` 目錄新增 `AddNewCard.vue` 檔案
- 將 `views/BoardView.vue` 的 `add new card` 區塊抽離至 `AddNewCard.vue`

```
<!-- views/BoardView.vue -->

<div class="bg-emerald-700 h-[100vh] w-full block overflow-x-auto overflow-y-hidden">
  <div id="board-wrapper" class="h-full w-full p-4 block overflow-auto">
    <div class="flex flex-row items-start">
      <!-- card -->
      <Card v-for="card in list" :key="card.id" v-bind="card" />

      <!-- add new card -->
      <AddNewCard />
    </div>
  </div>

  <!-- lightbox -->
  <EditBox v-if="currentEditTask?.id" />
</div>
```

新增卡片 (新增其他列表)

- 仿造前面的做法改寫 `AddNewCard.vue`

```vue
<script setup>
import { ref } from "vue";
import { useFocus } from "@vueuse/core";
import { vOnClickOutside } from "@vueuse/components";

// 處理使用者游標焦點
const target = ref();
useFocus(target, { initialValue: true });

// 切換狀態
const isEditing = ref(false);
const title = ref("");
</script>
```

新增卡片 (新增其他列表)

- 仿造前面的做法改寫 `AddNewCard.vue`

```
<template>
  <div
    class="border rounded-sm cursor-pointer bg-slate-200 bg-opacity-70 border-gray-500 mx-2 min-w-[300px] p-2 w-[300px]
  >
    <div @click="isEditing = true" v-if="!isEditing" class="block select-none">
      + 新增其他列表
    </div>
    <div v-else>
      <input
        type="text"
        ref="target"
        v-on-click-outside="() => (isEditing = false)"
        placeholder="為列表輸入標題"
        class="w-full p-2 block"
        v-model="title"
      />
    </div>
  </div>
</template>
```

新增卡片 (新增其他列表)

- 如果覺得 Tailwind CSS 的 Class 太長，可以用 `@apply` 來簡化

```
<template>
  <div class="new-card">
    <div @click="isEditing = true" v-if="!isEditing" class="block select-none">
      + 新增其他列表
    </div>
    <div v-else>
      <!-- 略 -->
    </div>
  </div>
</template>

<style scoped>
.new-card {
  @apply border rounded-sm cursor-pointer bg-slate-200 bg-opacity-70 border-gray-500
    mx-2 min-w-[300px] p-2 w-[300px] block hover:bg-opacity-90;
}
</style>
```

新增卡片 (新增其他列表)

- 到 `stores/index.js` 新增 `addNewCard` 方法

```
// 新增卡片
const addNewCard = (title = "") => {
  if (!title) return;

  lists.value.push({
    id: uid(),
    title,
    tasks: [],
  });
};
```

新增卡片 (新增其他列表)

- 回到 `AddNewCard.vue` ，新增 `addCard` ：

```
const store = useStore();
const { addNewCard } = store;

const addCard = () => {
  addNewCard(title.value);
  title.value = "";
  isEditing.value = false;
};
```

新增卡片 (新增其他列表)

- 同樣加上 `@keydown.enter` 事件

```
<input
  type="text"
  ref="target"
  v-on-click-outside="() => (isEditing = false)"
  @keydown.enter="addCard"
  placeholder="為列表輸入標題"
  class="w-full p-2 block"
  v-model="title"
/>
```

新增卡片 (新增其他列表)

- 完成。

加入拖拉排序

拖拉排序

- 這邊使用 vue.draggable.next 來實作拖拉排序
- 需要排序的部分
  - 卡片列表 `Card.vue`
  - 任務列表 `TaskItem.vue`

draggable

- `list` ：要排序的原始陣列
- `group` ：分組名稱，用來決定是否可以互相拖放
- `itemKey` ：用來判斷是否為同一個元素 (等同 `v-for` 的 `:key`)
- `ghost-class` : 拖拉時的樣式

```
<draggable :list="array" group="group-name" itemKey="id" ghost-class="opacity-30">
  <template #item="{ element }">
    <!-- 原本要排序的內容 -->
  </template>

  <template #footer>
    <!-- footer slot -->
  </template>
</draggable>
```

卡片排序

- 修改 `views/BoardView.vue`

```
<script setup>
import draggable from "vuedraggable";
// ...其他略
</script>
```

- 改寫這段：

```
<div class="flex flex-row items-start">
  <!-- card -->
  <Card v-for="card in list" :key="card.id" v-bind="card" />

  <!-- add new card -->
  <AddNewCard />
</div>
```

卡片排序

- 使用 `<draggable>` 元件包覆

```
<draggable
  :list="list"
  group="card"
  itemKey="id"
  ghost-class="opacity-30"
  class="flex flex-row items-start"
>
  <!-- 原本的 Card -->
  <template #item="{ element }">
    <Card v-bind="element" />
  </template>

  <!-- AddNewCard -->
  <template #footer>
    <AddNewCard />
  </template>
</draggable>
```

卡片排序


Hello Laravel!
Laravel 測試
內文區塊
Laravel 測試2
內文區塊2
+ 點擊以新增任務
Hello Vue.js!
Vue.js 測試
內文區塊
Vue.js 測試2
內文區塊2
+ 點擊以新增任務
+ 新增其他列表
Hello Vue.js!
Vue.js 測試
內文區塊
Vue.js 測試2
內文區塊2
+ 點擊以新增任務

任務列表排序

- 開啟 `components/Card.vue`

```
<!-- tasks 改寫前 -->
<TaskItem
  v-for="task in tasks"
  :key="task.id"
  @click="openEditTask(props.id, task.id)"
  v-bind="task"
/>
```

任務列表排序

- 改寫 `Card.vue` ，加上 `<draggable>` 元件包覆

```
import draggable from "vuedraggable";
```

```
<!-- tasks 改寫後 -->
<draggable :list="tasks" group="task" itemKey="id" ghost-class="opacity-30">
  <template #item="{ element }">
    <TaskItem
      @click="openEditTask(props.id, element.id)"
      v-bind="element"
    />
  </template>
</draggable>
```

任務列表排序

- 改寫完成


Hello Vue.js!
Vue.js 測試
內文區塊
+ 點擊以新增任務
Hello Laravel!
Laravel 測試
內文區塊
Vue.js 測試2
內文區塊2
Laravel 測試2
內文區塊2
+ 點擊以新增任務
+ 新增其他列表

利用 localStorage 將列表狀態持久化

列表狀態持久化

- 改寫 `stores/index.js` ，加上 `watch` 監聽 `lists` 的變化：

```
// 為 lists 加入 watch 監聽
watch(
  lists,
  (val) => {
    // 當 list 變動時，將變動後的值存入 localStorage
    localStorage.setItem("trello-lists", JSON.stringify(val));
  },
  { deep: true }
);
```

列表狀態持久化

- 改寫 `` `const lists = ref(defaultList);` ``
- 重整畫面後，由於列表狀態被保留在 localStorage，所以列表狀態不會被重置

```
export const useStore = defineStore("store", () => {
  const lists = ref(JSON.parse(localStorage.getItem("trello-lists")) || defaultList);

  // 下略
});
```

透過網址路由顯示看板

利用 Vue-Router 處理網址路由

- 在 `views/` 目錄下新增 `TaskView.vue`
- 並將原本在 `views/BoardView.vue` 的燈箱內容移至 `views/TaskView.vue`

```
<!-- views/TaskView.vue -->

<script setup>
import { computed } from "vue";
import { useStore } from "/src/stores";
const store = useStore();
const currentEditTask = computed(() => store.currentEditTask);
</script>

<template>
  <EditBox v-if="currentEditTask?.id" />
</template>
```

利用 Vue-Router 處理網址路由

- 原本的 `views/BoardView.vue` 的燈箱部分則由 `<router-view />` 來替代

```
<div class="bg-emerald-700 h-[100vh] w-full block overflow-x-auto overflow-y-hidden">
  <div id="board-wrapper" class="h-full w-full p-4 block overflow-auto">
    <draggable>
      <!-- 略 -->
    </draggable>
  </div>

  <!-- lightbox -->
  <router-view />
</div>
```

利用 Vue-Router 處理網址路由

- 修改 `router/index.js`

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: "/",
      component: () => import("/src/views/BoardView.vue"),
      children: [
        {
          path: "task/:cardId/:taskId",
          name: "task",
          component: () => import("/src/views/TaskView.vue"),
        },
      ],
    },
  ],
});
```

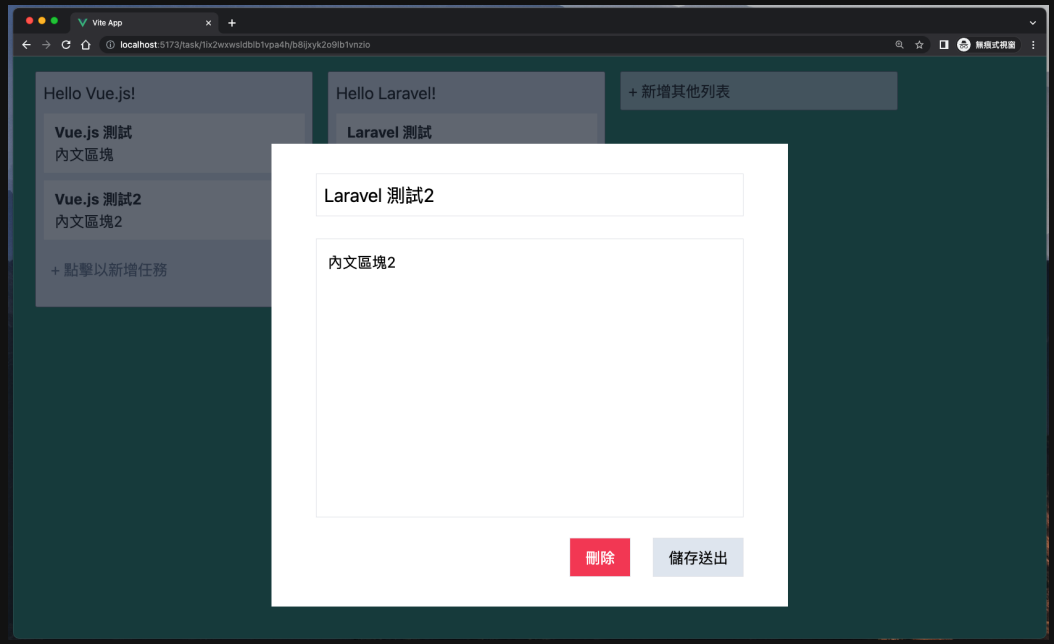
利用 Vue-Router 處理網址路由

- 修改 `stores/index.js` ，加上 `useRouter` 控制網址狀態

```
import { useRouter } from "vue-router";

export const useStore = defineStore("store", () => {
  const $router = useRouter();
  // ...略

  // 開啟編輯燈箱
  const openEditTask = (cardId, taskId) => {

    // ...其他部份省略，加入這行
    $router.push(`/task/${cardId}/${taskId}`);
  };

  // 清空 currentEditTask 代表關閉燈箱
  const closeEditTask = () => {
    currentEditTask.value = {};
    $router.push(`/`);
  };
});
```

利用 Vue-Router 處理網址路由

- 點擊任務卡片，會發現網址導向 `` `/task/:cardId/:taskId` `` 的路由
- 關閉後會回到 `` `/` ``



利用 Vue-Router 處理網址路由

- 改寫 `views/TaskView.vue`，讓使用者從指定的網址進入自動開啟燈箱

```
<script setup>
import { computed, onMounted } from "vue";
import { useStore } from "/src/stores";
import { useRoute } from "vue-router";

const store = useStore();
const currentEditTask = computed(() => store.currentEditTask);
const { openEditTask } = store;

const $route = useRoute();
const { params } = $route;
const cardId = computed(() => params.cardId);
const taskId = computed(() => params.taskId);

onMounted(() => {
  if (cardId.value && taskId.value) {
    openEditTask(cardId.value, taskId.value);
  }
});
</script>
```

完成！

成品原始檔: https://tinyurl.com/vue-trello