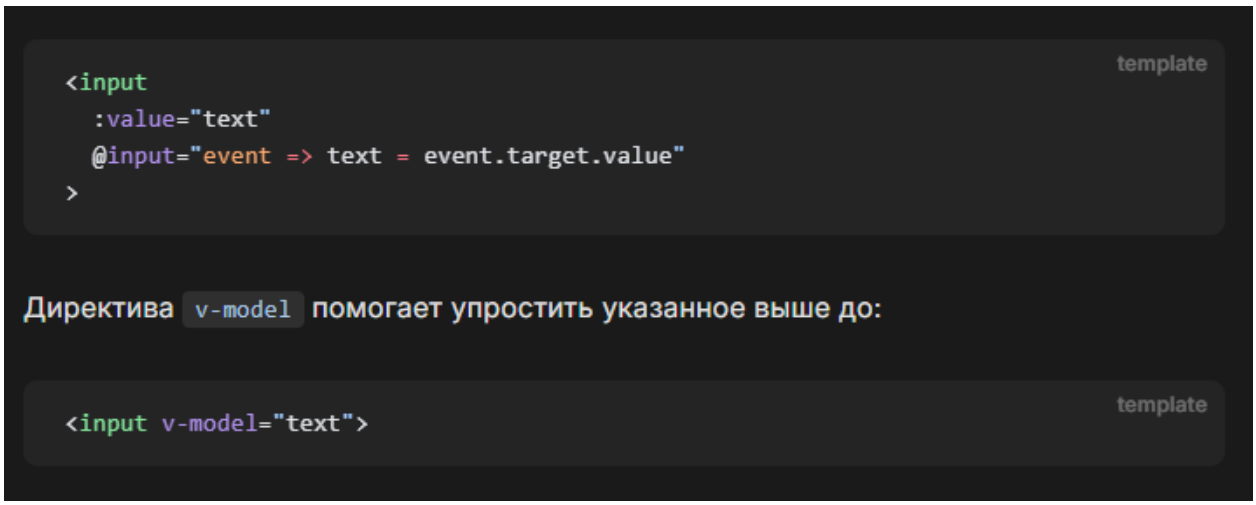
Дебаунсеры. Работа с HTML-формами. Двустороннее связывание состояния. Слоты.

В одном из прошлых заданий Вы написали обработчик события «input» для ввода поискового запроса по заметкам. Если Вы откроете DevTools на вкладке «Network», то увидите, что при вводе каждого отдельного символа в строку поиска происходит отправка запроса, что сильно нагружает сервер. Чтобы избежать лишней нагрузки на сервер, были придуманы функции-дебаунсеры (обычно они имеют название «debounce»).

Дебаунсер принимает коллбэк в качестве одного из своих параметров и ограничивает кол-во его вызовов путём установки таймаута величиной N секунд (или миллисекунд, в зависимости от задачи). В том случае, если далее вызовов дебаунсера не последовало, то коллбэк вызовется через N секунд. Но если дебаунсер был вызван до того момента, пока не прошло N секунд, то отсчёт N секунд начинается заново и коллбэк всё это время ждёт вызова. Такой функционал оказывается очень полезным при реализации его на обработчиках событий ввода в поисковую строку и другие подобные фильтры.

В статье [1] даётся подробный обзор функций дебаунсеров, обосновывается их необходимость и представлен способ реализации.

Теперь немного о двустороннем связывании состояний. При реализации строки поиска Вы скорее всего привязали к элементу <input> атрибут value и событие input. В функции обработчике события input Вы изменяли значение атрибута value посредством ref-ссылки, используя реактивность Vue. Однако, в случае, если Вам не требуется выполнить каких-либо побочных операций по отношению к перечисленным, Вы можете упростить синтаксис и воспользоваться директивой v-model на компоненте <input>, передав ей аргумент ref-ссылки. Ниже представлен скриншот из документации [2]:



В зависимости от компонента директива v-model разворачивается в соответствующие пары «значение-событие». Например, в <textarea> это будет «value, input», а в <input type="checkbox"> – «checked, change». Более подробно в документации [2].

При создании компонента модального окна Вам понадобится реализовать его так, чтобы его можно быть использовать с различным содержимым внутри. Причём содержимое есть не что иное, как некоторый шаблон (DOM-поддерево) – в нашем случае это HTML-форма. То есть требуется передать в компонент не только свойства (props), но и шаблон.

В фреймворке Vue это можно реализовать через специальный тэг <slot>, который указывается в родительском компоненте (в нашем случае это будет само модальное окно без содержимого, точнее единственным содержимым будет тэг <slot>). Теперь, когда мы передадим в качестве дочернего элемента (HTML-форму) в родительский (модальное окно) некоторое поддерево, то тэг <slot> будет заменён переданным поддеревом. Таким образом, мы можем переиспользовать родительский компонент много раз, меняя его содержимое в зависимости от ситуации.

Слотов в компоненте может быть указано сколько угодно, причём они могут быть именованными, что удобно в некоторых сценариях. Также на них можно повесить директивы условной компиляции v-if и v-else. Все возможные примеры использования слотов представлены в [5].

Вопросы и задания

1. Продолжаем работу с приложением notes-app. Реализуйте composable-функцию дебаунсера в директории /src/composables, назвав её «useDebounceFn». Установите время таймаута, равное 250 мс. Примените реализованную функцию в основной части приложения для строки поиска.

2. Изучите макет [3] модального окна с формой создания / редактирования заметки. Так как модальное окно с его свойствами (затемнение фона, закрытие при клике на затемнение и т. д.) может быть переиспользовано в дальнейшем, создайте глобальный SFC-компонент c-modal.vue в директории /src/components/c-modal. Если такой директории ещё нет в проекте, создайте новую. Желательно воспользоваться HTML-тегом <dialog>, так как он семантически больше подходит для данной задачи. В качестве полезного содержимого модального окна должен выступать <slot>. Стилизуйте согласно дизайну и добавьте функционал закрытия.

3. Создайте локальный SFC-компонент формы создания / редактирования заметки согласно макету [3]. Расположите его в директории /src/views/notes/components/c-note-edit-form. Не забудьте про семантику HTML-элементов и выберите необходимый корневой тэг.
   Заголовок формы должен отличаться в зависимости от контекста вызова: если выбрано действие создания новой заметки, то «Новая заметка», если выбрано редактирование существующей заметки, то «Редактирование заметки». Во втором случае поля должны быть предзаполнены для выбранной заметки (кроме поля выбора файла).

   Для заголовка заметки используйте тэг <input>, а для текста – <textarea>.

   Добавьте валидацию для полей ввода:

   - строка заголовка не более 10 символов
   - строка текста не более 1000 символов
   - загрузка файла не более 1 МБ и только форматы .jpeg, .jpg, .png.

4. Выведите модальное окно в связке с формой создания / редактирования заметки в компонент /src/views/notes.vue. Напишите

запросы с использованием функции useRequest для создания и редактирования заметки. Соответствующие эндпоинты указаны в файле README.md [4].

Воспользуйтесь статусами запросов для отображения прелоадера на кнопке действия формы. Например, пока ожидается ответ от сервера, можно на кнопке «Сохранить» формы редактирования вместо текста отобразить троеточие «…» и вывести в состояние «disabled», а кнопку «Отмена» просто вывести в состояние «disabled» (см. макеты [3]). После получения ответа модальное окно должно быть закрыто.

Ссылки

[1]. https://doka.guide/js/debounce/

[2]. https://ru.vuejs.org/guide/essentials/forms.html

[3]. https://www.figma.com/design/YuGxLRvd4ll72xl8f96wbu/%D0%A8%D0%BA%D0%BE%D0%BB%D0%B0_frontend_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%B0?node-id=0-1&p=f&m=dev

[4]. https://github.com/GreysMouse/notes-app-server

[5]. https://ru.vuejs.org/guide/components/slots.html