Ход выполнения работы:

<https://www.youtube.com/watch?v=DATk9QsSRTQ&t=138s&ab_channel=%D0%9F%D0%B0%D0%B2%D0%B5%D0%BB%D0%91%D0%BE%D1%80%D0%B8%D1%81%D0%BE%D0%B2>

описание пропсов, добавление классов, интерполяция: 18 минута

21:15

массив задач в поле data в export default deifneComponent

для v-for в todolist

далее (24 минута) опишем интерфейс для возвращаемого значения функции data (который уже есть в AppTodoItem) - чтобы нельзя было создать пустой задачи, чтобы были нужные поля с правильными типами

это уже typescript

итак, функция data возвращает объект типа State (интерфейс)

27 минута: мы тот же тип указываем в двух местах, значит, надо вынести в отельный тип - в src теперь будет папка types, из файла будем экспортировать интерфейс Todo (замена в двух компонентах)

теперь нужно разобраться с переключением задач (их состояния и т.д) - оживить интерфейс

научим карточку **менять состояние** по клику - ставим слушатель клика на 29 минуте

добавляем секцию методов в defineComponent

в методе тогл хочется просто ставить completed = !completed

но пропы нельзя мутировать

нам нужно заэммитить туду наверх, пусть с этим разбирается компонент, который эту туду и предоставил (мы сейчас в Item)

на 31.30 минуте типизируем emit

на 32.30 вместо массива типизируем emit в объектном стиле - чтобы данные эмита приходили в верном формате

на 34.30 переходим в AppTodoList и прослушаем это событие - добавляем toggle-todo

на 36.30 описываем логику метода (вместо переключения флага как изначально хотели)

**Удаление задачи**: вешаем событие клик на кнопку с классом remove-button в AppTodoItem, будем вызвать метод removeTodo - по сути копию метода toggleTodo - однако из-за "вложенности" @click-ов нужно добавить .stop - чтобы один метод не вызвал другой

далее делаем аналогично прошлому в AppTodoList, но внутри с измененной логикой: запишем отфильтрованный массив todos, из которого уберем id задачи, который получили

41 минута видео

Делаем добавление задачи, снова используем интерфейс State

Добавляем к элементу <form> v-if

далее спускаем кнопку под форму с v-else

далее навешиваем слушатель клика на эту кнопку - добавляем метод showForm в defineComponent, этот метод ничего не принимает, а просто меняет флаг на значение true

теперь сделаем так, чтобы клик по крестику закрывал **форму добавления задачи**

Теперь пишем **логику добавления задачи**

пользователь влияет только на название задачи - но не на ее id (генерируется автоматически) или флаг компонента (он по умолчанию false)

поэтому в AppAddTodo в defineComponent в поле data заведем поле todoText (текст, который пользователь вводит в инпут) - по умолчанию оно пустое

и добавим это поле в State

далее добавим с помощью v-model двустороннее связывание на этот инпут (с полем todoText)

нужно установить vue devtools

и проверить компонент AppAddTodo

все ок

следующее что должны сделать - по нажатию кнопки add task отправлять компонент todo выше - в компонент App

46.40

на саму кнопку вешать слушатель не будем, т.к это вызовет проблемы, повесим слушатель события submit на форму

будет модификатор prevent, который предотвратит перезагрузку страницы при отправке формы

на слушатель повесим метод addTodo

метод addTodo будет эмитить наверх с передачей объекта - задачи (id будет по текущему времени, т.е уникальным)

и опишем секцию emits - функция валидации и тип Todo (не забыть импорт)

теперь идем в компонент App.vue и навешиваем слушатель, ну и описание метода в defineComponent, как обычно

и не забыть про импорт типа Todo

метод пока выводит объект в консоль

теперь надо **добавить объект в список задач на странице**, только перед этим нужно очистить поле ввода - 50.00

Очистим поле ввода (вернемя в AppAddTodo.vue) - после эмита сбросим поле ввода: this.todoText=''

итак, как добавить полученный объект в список?

проще всего будет вынести массив задач и методы из компонента AppTodoList (**разгрузим** его, он будет заниматься только отрисовкой) в компонент App

переносим поле data, не забываем перенести интерфейс State, переносим методы

(пока ctrl c + ctrl v)

теперь в App.vue компонент AppTodoList, из data которого ушли задачи, принимал их в качестве пропа todos="todos"

далее в AppTodoList удаляем data и интерфейс State (но импорт Todo остается), очистим методы, чтобы они не мутировали проп todos - методам останется только эмитить дальше эти события - 54.00

и опишем в AppTodoList пропс todos - это массив as Proptype<Todo[]> - не забываем про импорт PropType

и не забываем описать emits в AppTodoList (они идентичны тем, что в AppTodoItem)

55.00

Возвращаемся в компонент App и на компонент AppTodoList вешаем слушатели событий @toggle-todo и @remove-todo, эти методы уже описаны внизу (те, что были перенесены из AppTodoList)

Объяснение подробнее - на 55.20

теперь AppTodoList ничего не решает, просто принимает на вход тудушки в массив задач и отрисовывает их. События ловит и прокидывает дальше в App - вся логика по тоглу и удалению в App.vue (сейчас добавим и удаление)

т.е займемся методом addTodo (заменим console.log на нормальный код - this.todos.push(todo))

готово! 56.30

57.00 - осталось **реализовать фильтрацию и статистику**

Начнем с фильтрации, перейдем в компонент AppFilters.vue - там набор кнопок, активная кнопка подсвечивается красным

будем передавать и получать значение фильтра в пропах: тип пропа будет строка, которая принимает значения All, Active, Done, и в зависимости от этого меняется класс кнопки - для этого чтобы строка не была любой, а принимала только одно из трех значений, заведем PropType (не забудем про его импорт) - этот проп будет обязательным (required), т.к мы всегда ожидаем этот фильтр

59.30

1.00.30

перейдем в компонент App

внутри него компонент AppFilter ожидает обязательный фильтр - добавим его пока вручную

двойные кавычки " 'Active' ", например

но вручную мы прокидывать, конечно, не будем

и нам в State понадобится значение этого фильтра, чтобы фильтровать список задач

добавим в State activeFilter c тремя доступными значениями

теперь этот код с тремя доступными значениями повторяется в двух местах - в App.vue и в AppFilters.vue - вынесем его в отдельный файл Filter.ts в папке types (экспортируем в нем тип)

1.02.30

перейдем в AppFilters и заменим тип для фильтра

не забыть про импорт файла!

в App.vue сделаем то же самое

и добавим в data activeFilter со значением по умолчанию 'All'

1.03.00

Теперь компоненту AppFilters в начале App.vue передадим activeFilter (а не вручную, как в начале)

Возвращаемся в компонент AppFilters и доделаем вторую половину

Повесим событие клика на каждый фильтр, чтобы эмитить тип фильтра, по которому произошел клик

на каждый фильтр вешать клик...

поэтому лучше отрисовать фильтры сразу же в цикле

поэтому заведем поле data в AppFilters.vue, в котором будет поле массив фильтров

и поэтому опишем State с фильтрами

1.05.20

В AppFilters.vue удаляем все кнопки, кроме одной, и будем отрисовывать их в цикле v-for

применяем key - уникальный ключ, заменяем условие в тернарном операторе, применяем интерполяцию {{ }}

1.06.00

и вот теперь добавляем слушатель события клик (который будет эмитить наверх) в метод нужно передать фильтр!

1.07.25

возвращаемся в компонент App.vue и в компоненте AppFilters прослушаем событие @set-filter и опишем метод setFilter в methods

1.08.00

после получения значения фильтра необходимо отфильтровать список задач

будем делать это в App.vue в computed properties (поле computed)

новое вычисляемое свойство - filteredTodos, оно возвращает массив Todo[]

внутри - логика для фильтрации через switch-case

в кейсе All и default мы отображаем все задачи, поэтому избавляемся от дублирования кода

case 'All':

default:

return this.todos

теперь используем вычисляемое свойство

использовать будем в компоненте <AppTodoList /> - прокидываем вместо массива тудушек отфильтрованный массив тудушек

1.11.00

остался последний штрих - **дописать секцию со статистикой**

она выводит два числа - количество задач в работе и количество выполненных задач

переходим в AppFooter.vue и научим принимать его проп со статистикой stats, опишем вариант передачи пропа в экспортируемом интерфейсе Stats (+ делаем импорт PropType)

т.к проп обязателен, ставим required: true

1.12.30

теперь через интерполяцию выводим эти значения (в AppFooter.vue)

переходим в компонент App.vue

нужно передать этот объект со статистикой в <AppFooter />

поскольку все вычисляется в реальном времени, добавим в computed свойство stats с импортом интерфейса Stats

и скопируем в него логику вычисляемого свойства filteredTodos, только нам нужна длина массива, поэтому дописываем .length

и это свойство stats в App.vue, которое только что описали, должно передаваться в <AppFooter />

<AppFooter :stats="stats" />

на этом можно закончить, но для чистоты кода без повторений вынесем в computed-свойства activeTodos и doneTodos, и будем к ним обращаться в filteredTodos и stats через this