Начало работы над приложением Todo list

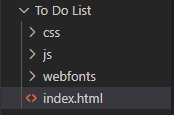
Создать папку проекта.

Файл index.html

Папка css – файлы из bootrstrap/css и fontawesome/css

Папка js – файлы из bootrstrap/js и vue.js

Папка webfonts из fontawesome



Далее создаем базовый html

Изображение выглядит как текст

Автоматически созданное описание

Здесь мы подключили стили bootstrap, иконки fontawesome, скрипты bootstrap и vue.

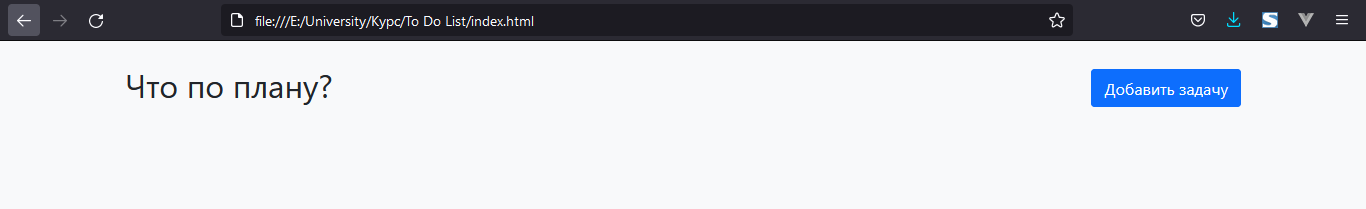
Также создали элемент #app, куда рендерится приложение.

Зададим body фон с помощью класса bg-light.

Создаем базовый контейнер с вертикальным отступом, заголовок и кнопку добавить. Используем классы bootstrap:

Изображение выглядит как текст, экран, снимок экрана, набор

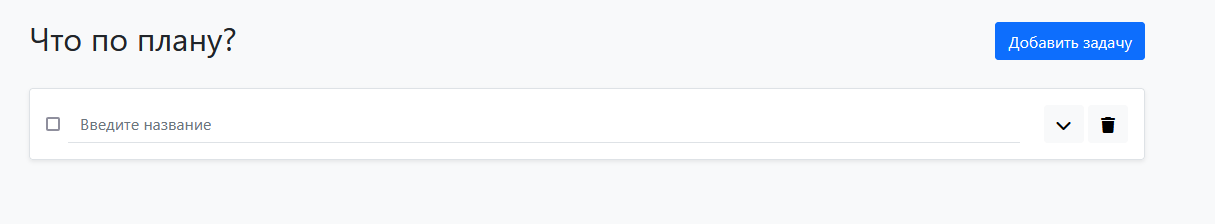
Автоматически созданное описание



После заголовка и кнопки сделаем элемент задачи. Здесь как раз используем иконки.

Изображение выглядит как текст

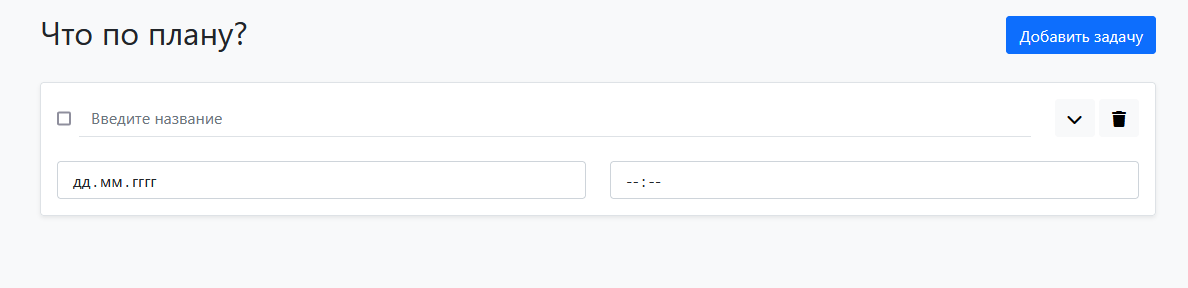
Автоматически созданное описание



Добавим форму редактирования даты и времени

Изображение выглядит как текст

Автоматически созданное описание



А теперь сделаем, чтобы эта форма была раскрывающейся. Модифицируем код кнопки со стрелкой так, добавив класс collapsed и атрибуты data-bs-toggle и data-bs-target.

Изображение выглядит как текст

Автоматически созданное описание

И блока с формой – добавили класс collapse и id=”task-1”:

Изображение выглядит как текст

Автоматически созданное описание

Здесь использовали Bootstrap collapse. Атрибут data-target указывает, какой элемент триггерит кнопка. В данном случае это элемент #task-1. Подробнее, как это работает, можно почитать по ссылке <https://getbootstrap.com/docs/5.0/components/collapse/>

Добавим стиль для стрелки, чтобы она крутилась. Создадим файл custom.css в папке css и подключим. Добавим такой код:

Изображение выглядит как текст

Автоматически созданное описание

Кнопка при открытии блока теряет класс collapsed. За этим можно понаблюдать, переключая ее.

**С версткой пока закончили.**

Создаем объект Vue:

Изображение выглядит как текст

Автоматически созданное описание

Пока из данных у нас пустой массив задач. Скоро задач прибавится.

Сразу создадим методы добавления и удаления, потом приделаем их к кнопкам:

Изображение выглядит как текст

Автоматически созданное описание

Метод addTask добавляет В НАЧАЛО МАССИВА объект с полями completed, title, date, time.

Метод deleteTask удаляет из массива по индексу.

Теперь привяжем массив к элементам верстки. Используем директиву v-for:

Изображение выглядит как текст

Автоматически созданное описание

Здесь главному блоку добавили вывод по циклу, который пробегается по всем элементам массива:

v-for="(task, index) in tasks" :key="index"

полям ввода сразу назначили v-model:

v-model="task.completed"

v-model="task.title"

v-model="task.date"

v-model="task.time"

И подвязали collapse через динамические атрибуты. У нас в примере было task-1, здесь мы динамически подставим индекс, чтобы кнопка соответствовала блоку.

data-bs-toggle="collapse" :data-bs-target="`#task-${index}`"

class="collapse" :id="`task-${index}`"

*\*\*можно было лепить строки и так:*

*:id="'task-' + index"*

*Но мы будем использовать шаблон строк, так как он более удобный и меньше вероятность ошибиться с кавычками. Синтаксис получается такой:*

*let str = `три плюс два равно ${ 3 + 2}`*

*Подробнее* [*https://dev-gang.ru/article/-sposoba-obedinenija-strok-v-javascript-prgtacj4yl/*](https://dev-gang.ru/article/-sposoba-obedinenija-strok-v-javascript-prgtacj4yl/)

Сейчас у нас ни одного элемента нет, ведь массив пуст. Правда, в начале может мелькать на странице. Добавим блоку #app атрибут v-cloak:

<div id="app" v-cloak>

И в custom.css добавим:

\*[v-cloak] {

    display: none;

}

Теперь элемент #app будет видим только после отрисовки контента.

Вывод списка есть, надо научить добавлять задачи. Повесим обработчик на кнопку:

Изображение выглядит как текст

Автоматически созданное описание

Добавили

@click="addTask"

Теперь при нажатии на кнопку добавляются новые пункты:

Изображение выглядит как текст

Автоматически созданное описание

Сделаем, чтобы не прилипали. Добавим класс .mb-3:

<div class="border bg-white shadow-sm rounded p-3 mb-3" v-for="(task, index) in tasks" :key="index">

Стало лучше:

Изображение выглядит как текст

Автоматически созданное описание

Теперь реализуем удаление по кнопке корзины:

Изображение выглядит как текст

Автоматически созданное описание

Добавили

@click="deleteTask(index)"

Теперь можно добавлять и удалять.

Добавим иконку для кнопки Добавить:

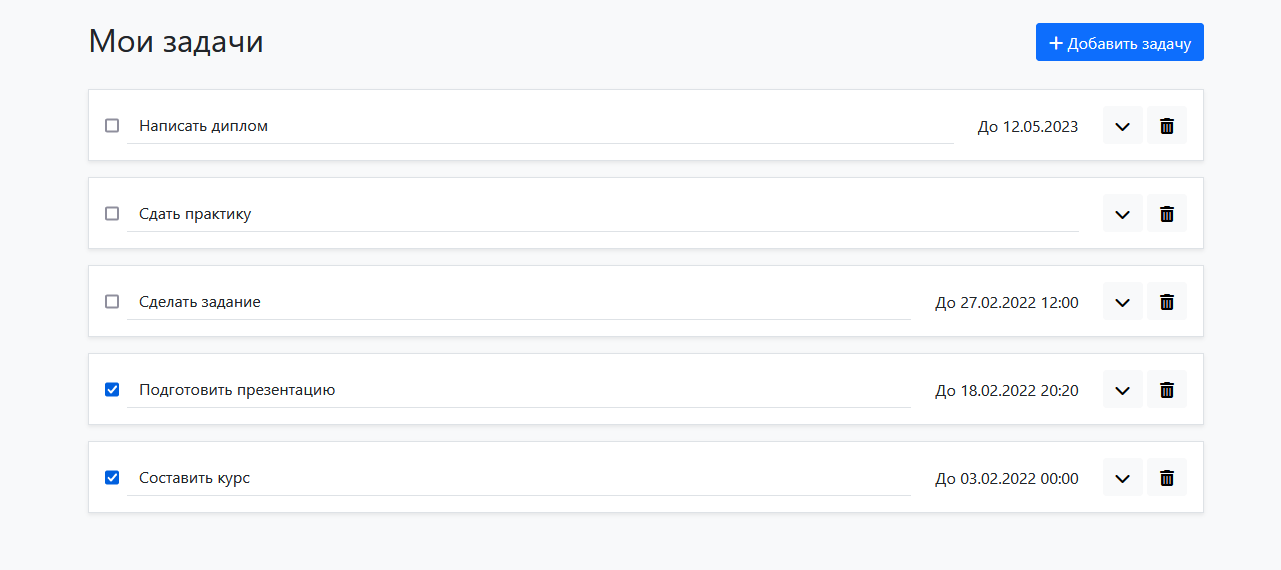
Изображение выглядит как текст

Автоматически созданное описание

После блока .col добавим блок .col-auto для отображения даты и времени, если они указаны. Дата приводится к нормализованному виду.

Изображение выглядит как текст

Автоматически созданное описание



Теперь сделаем текст красным, если задача просрочена. Используем динамические классы:

Изображение выглядит как текст

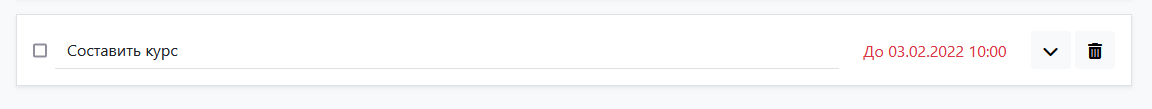
Автоматически созданное описание

Класс text-danger будет добавлен, если checkTime возвращает true. Напишем этот метод:

Изображение выглядит как текст

Автоматически созданное описание

Он сверяет дату дедлайна по задаче с текущей. При этом, если дата не назначена, берется текущая. Обратите внимание, getMonth() возвращает номера месяцев, начиная с нуля.



Теперь сделаем автосохранение. Напишем метод-наблюдатель:

Изображение выглядит как текст

Автоматически созданное описание

При изменении даже внутренних свойств данные сериализуются в JSON и сохраняются в локальное хранилище браузера.

И загрузка при первоначальной отрисовке:

Изображение выглядит как текст

Автоматически созданное описание

Теперь при перезагрузке страницы данные должны оставаться.

Теперь сверстаем форму поиска после блока с заголовком и кнопкой:

Изображение выглядит как текст, снимок экрана, экран

Автоматически созданное описание

И добавим свойство searchText, которое связано с этим полем:

Изображение выглядит как текст

Автоматически созданное описание

Поиск реализуем с помощью вычисляемого свойства filteredTasks. Создадим его:

Изображение выглядит как текст

Автоматически созданное описание

Это массив, отфильтрованный по условию: название содержит строку запроса, при этом сравнение без учета регистра.

Еще переделаем логику с ключами, чтобы индексы работали правильно. Заведем поле id при создании задачи. Чтобы оно было уникальным, будем присваивать ему текущую дату в мс).

ПРЕДВАРИТЕЛЬНО УДАЛИМ ВСЕ ЗАДАЧИ и перепишем метод addTask и метод deleteTask:

Изображение выглядит как текст

Автоматически созданное описание

В шаблоне заменим tasks на filteredTasks и заменим index на task.id:

Изображение выглядит как текст

Автоматически созданное описание

Теперь можно искать задачи по названию:

Изображение выглядит как текст

Автоматически созданное описание

Сделаем отдельный блок для выполненных задач. Добавим еще 2 вычисляемых свойства activeTasks и completedTasks:

Изображение выглядит как текст

Автоматически созданное описание

Это будут активные и выполненные задачи.

Добавим в шаблон блок сразу под циклом с задачами:

Изображение выглядит как текст

Автоматически созданное описание

Теперь, если есть выполненные задачи, то они будут отображаться ниже:

Изображение выглядит как текст

Автоматически созданное описание

Доработаем UX. Обработаем случай, когда ни одной таски нет. В этом случае отображаем какой-нибудь текст. Добавим его после блоков с тасками:

Изображение выглядит как текст

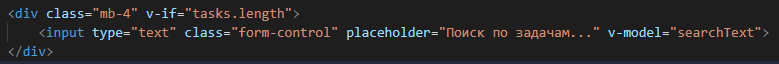
Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Можно, в принципе, и не показывать поле поиска, если ничего нет. Добавим v-if:



Изображение выглядит как текст

Автоматически созданное описание

Теперь что если задачи есть, но под условия поиска ни одна не подходит? Добавим блок с условием, что задачи есть, но нет (tasks.length > 0 и filteredTasks == 0). Шаблон примет следующий вид:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

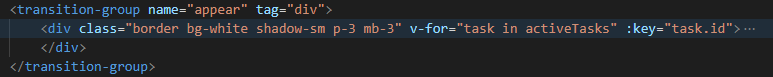
Автоматически созданное описание

Еще сделаем стирание строки поиска при добавлении нового элемента. Потому что сейчас, если у нас задано условие поиска и мы добавляем новый элемент, мы его не видим, что неудобно юзеру.

Изображение выглядит как текст

Автоматически созданное описание

Теперь добавим анимацию списка. Обернем цикл по блокам в элемент <transition-group> и сделаем ему атрибуты name=”appear” и tag=”div”:



И добавим стили в CSS:

Изображение выглядит как текст

Автоматически созданное описание

Теперь добавление и удаление должно быть анимировано. Аналогично пропишем анимацию списку выполненных задач:

Изображение выглядит как текст

Автоматически созданное описание