Todolist_React_yup_formik

Задание: Создание компонента "Умный список задач" на React с TypeScript

Общая цель: Разработать полнофункциональное приложение для управления персональными задачами с валидацией, интерактивными элементами и визуальным отображением статусов.

📋 ПОДГОТОВИТЕЛЬНЫЙ ЭТАП: ИМПОРТЫ И НАСТРОЙКА

Шаг 1.1: Импорт React функционала

- Импортируй основной хук для управления внутренним состоянием компонента
- Этот хук позволит хранить и обновлять массив задач

Шаг 1.2: Импорт библиотеки для работы с формами

- Импортируй специальный хук для упрощения работы с формами
- Этот хук предоставляет готовые методы для управления значениями полей, валидации и отправки

Шаг 1.3: Импорт библиотеки валидации

- Импортируй всю библиотеку валидации под удобным псевдонимом

- Эта библиотека позволяет описывать правила проверки данных в декларативном стиле

Шаг 1.4: Импорт стилей

- Импортируй CSS-модуль для компонента под осмысленным именем
- CSS-модули обеспечивают локальную область видимости стилей

🎯 ОПРЕДЕЛЕНИЕ ТИПОВ ДАННЫХ

Шаг 2.1: Создание типа задачи

- Определи пользовательский тип для представления отдельной задачи
- Тип должен содержать:
  - Уникальный числовой идентификатор
  - Текстовое описание задачи (обязательная строка)
  - Логический флаг, указывающий на статус выполнения

⚙️ ОСНОВНОЙ КОМПОНЕНТ И СОСТОЯНИЕ

Шаг 3.1: Создание функционального компонента

- Объяви функциональный компонент с понятным именем, который экспортируется по умолчанию
- Компонент не принимает пропсов

Шаг 3.2: Инициализация состояния задач

- Используй хук состояния для создания переменной состояния массива задач
- Начальное значение - пустой массив
- Укажи точный TypeScript-тип для массива задач

🔍 СИСТЕМА ВАЛИДАЦИИ

Шаг 4.1: Создание схемы валидации

- Используй метод объекта валидации для создания схемы

- Опиши правила для поля "text":
  - Значение должно быть строкой
  - Автоматическое обрезание пробелов в начале и конце
  - Минимальная длина 3 символа с соответствующим сообщением об ошибке
  - Максимальная длина 50 символов с соответствующим сообщением об ошибке
  - Обязательное поле с сообщением о необходимости ввода

📝 НАСТРОЙКА УПРАВЛЕНИЯ ФОРМОЙ

Шаг 5.1: Конфигурация формы

- Используй хук для работы с формами, передав объект конфигурации:
  - Начальные значения формы: поле "text" как пустая строка
  - Подключи созданную схему валидации
  - Опиши функцию, которая выполняется при отправке формы

Шаг 5.2: Логика обработки отправки формы

- Функция должна принимать значения формы и методы для сброса
- Внутри функции:
  - Создай новый объект задачи со следующими свойствами:
    - Уникальный ID на основе текущего времени
    - Текст задачи (обрезанный от пробелов)
    - Статус выполнения - false (не выполнена)
  - Обнови состояние задач, добавив новую задачу в конец массива
  - Сброси форму к начальным значениям

🔄 ФУНКЦИИ УПРАВЛЕНИЯ ЗАДАЧАМИ

Шаг 6.1: Функция переключения статуса выполнения

- Создай функцию, которая принимает ID задачи
- Функция должна обновлять состояние задач:
  - Пройдись по массиву задач
  - Найди задачу с соответствующим ID
  - Инвертируй значение свойства "completed"
  - Верни обновленный массив

Шаг 6.2: Функция удаления задачи

- Создай функцию, которая принимает ID задачи для удаления
- Функция должна обновлять состояние задач:
  - Отфильтруй массив, оставив только задачи, ID которых не совпадает с переданным
  - Верни новый отфильтрованный массив

🎨 ПОСТРОЕНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Шаг 7.1: Основная структура

- Создай корневой div-контейнер с применением стиля-обертки

Шаг 7.2: Заголовок приложения

- Добавь заголовок первого уровня с эмодзи списка и названием приложения
- Примени соответствующий стиль заголовка

Шаг 7.3: Форма ввода новой задачи

- Создай форму с обработчиком отправки и примени стиль формы
- Внутри формы размести:
  - Поле ввода текста с:
    - Типом "text"
    - Именем "text" (соответствует имени в Formik)

        - Плейсхолдером с приглашением к вводу
        - Значением из состояния формы
        - Обработчиком изменения значения
        - Обработчиком потери фокуса
        - Стилем для поля ввода
    - Кнопку отправки формы с:
        - Типом "submit"
        - Текстом "Добавить"
        - Стилем для кнопки добавления

Шаг 7.4: Блок отображения ошибок валидации

- Создай условный рендеринг для сообщений об ошибках:
    - Показывай сообщение только если поле было активировано (touched) И есть ошибка
    - Отображай текст ошибки из состояния формы
    - Примени стиль для сообщений об ошибках

Шаг 7.5: Список задач

- Создай ненумерованный список с применением стиля списка

Шаг 7.6: Сообщение о пустом списке

- Реализуй условный рендеринг:
    - Если массив задач пустой, покажи параграф с сообщением "Список пуст"
    - Примени соответствующий стиль для пустого состояния

Шаг 7.7: Рендеринг отдельных задач

- Для каждой задачи в массиве создай элемент списка с:
    - Уникальным ключом на основе ID
    - Стилем элемента списка

Шаг 7.8: Элемент задачи (текст)

- Внутри каждого элемента списка создай span для текста задачи:
  - Добавь обработчик клика для переключения статуса выполнения
  - Динамически применяй классы:
    - Базовый стиль текста задачи
    - Дополнительный стиль для завершенных задач (условно)
  - Отображай текст задачи

Шаг 7.9: Кнопка удаления задачи

- Рядом с текстом задачи размести кнопку:
  - Тип кнопки - "button" (чтобы не вызывать отправку формы)
  - Обработчик клика для удаления задачи
  - Символ "×" как содержимое
  - Стиль для кнопки удаления