🧪 Лабораторная работа

Разработка ToDo List на React + TypeScript + Vite + Formik + Yup + UUID

📌 Общая информация

Тип работы: лабораторная

Уровень: начальный / средний

Формат: пошаговая с подробными пояснениями

Результат: полнофункциональный ToDo List

🎯 Цели лабораторной работы

После выполнения лабораторной работы студент сможет:

- создать React-приложение с помощью Vite
- понимать структуру проекта React + TypeScript
- создавать функциональные компоненты
- работать с формами через Formik
- выполнять валидацию с помощью Yup
- управлять массивом объектов в состоянии React
- корректно изменять и удалять элементы списка
- использовать UUID как уникальный идентификатор
- соблюдать принцип иммутабельности состояния

🧰 Используемые технологии

- React

- TypeScript
- Vite
- Formik
- Yup
- uuid

🔷 ЭТАП 1. Создание проекта

🎯 Цель

Создать базовый React-проект с TypeScript.

🛠️ Терминал

```
npm create vite@latest todo-formik-lab -- --template react-ts
cd todo-formik-lab
npm install
npm run dev
```

📁 Структура проекта (упрощённо)

```
src/
├─ App.tsx
├─ main.tsx
└─ components/
```

✅ Результат

Проект запускается и отображается в браузере.

🔷 ЭТАП 2. Установка библиотек

🎯 Цель

Добавить библиотеки для форм, валидации и уникальных идентификаторов.

🛠️ Терминал

```
npm install formik yup uuid
```

❓ Пояснение

- **Formik** — управление состоянием формы
- **Yup** — декларативная валидация
- **uuid** — генерация уникальных идентификаторов

🔷 ЭТАП 3. Настройка корневого компонента

📄 Файл

`src/App.tsx`

🛠️ Код

```
import TodoForm from'./components/TodoForm';

functionApp() {
return (
<div>
<h1>ToDo List</h1>
<TodoForm />
</div>
```

```
  );
}

export default App;
```

❓ Пояснение

- `App` — корневой компонент
- Вся логика ToDo вынесена в `TodoForm`

🔷 ЭТАП 4. Создание компонента TodoForm

📁 Папка

`src/components`

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Базовая структура

```
const TodoForm = () => {
return (
<form>
<div>
<label>Название задачи</label>
<input type="text" />
</div>

<div>
<label>Описание задачи</label>
<textarea />
```

```
</div>

<button type="submit">Добавить задачу</button>
</form>
  );
};

export default TodoForm;
```

✅ Результат

Форма отображается, но пока не управляется React.

🔷 ЭТАП 5. Подключение Formik

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Импорт

```
import { useFormik }from'formik';
```

🛠️ Инициализация Formik

```
const formik =useFormik({
initialValues: {
title:'',
description:'',
  },
onSubmit:(values) ⇒ {
console.log(values);
```

```
  },
});
```

🛠️ Привязка формы

```
<form onSubmit={formik.handleSubmit}>
```

🛠️ Привязка полей

```
<input
type="text"
  name="title"
  value={formik.values.title}
  onChange={formik.handleChange}
  onBlur={formik.handleBlur}
/>

<textarea
name="description"
value={formik.values.description}
onChange={formik.handleChange}
onBlur={formik.handleBlur}
/>
```

❓ Пояснение

- Formik хранит значения формы
- `name` связывает input с Formik
- `handleSubmit` перехватывает submit

🔷 ЭТАП 6. Добавление Yup-валидации

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Импорт Yup

```
import * as Yup from'yup';
```

🛠️ Схема валидации

```
const validationSchema =Yup.object({
title:Yup.string()
    .min(3,'Минимум 3 символа')
    .required('Название обязательно'),
description:Yup.string()
    .required('Описание обязательно'),
});
```

🛠️ Подключение схемы

```
const formik = useFormik({
initialValues: {
title:'',
description:'',
  },
  validationSchema,
onSubmit:(values) => {
console.log(values);
  },
```

```
});
```

🔷 ЭТАП 7. Отображение ошибок

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Код

```
{formik.touched.title && formik.errors.title && (
<div style={{color: 'red' }}>
   {formik.errors.title}
</div>
)}

{formik.touched.description && formik.errors.description && (
<div style={{color: 'red' }}>
   {formik.errors.description}
</div>
)}
```

❓ Пояснение

Ошибки отображаются только после взаимодействия с полем.

🔷 ЭТАП 8. Добавление состояния списка задач

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Импорты

```
import { useState } from'react';
import { v4 as uuidv4 } from'uuid';
```

🛠️ Тип задачи

```
typeTodo = {
id:string;
title:string;
description:string;
completed:boolean;
};
```

🛠️ Состояние

```
const [todos, setTodos] = useState<Todo[]>([]);
```

🛠️ Обновление onSubmit

```
onSubmit:(values, { resetForm }) ⇒ {
const newTodo:Todo = {
id:uuidv4(),
title: values.title,
description: values.description,
completed:false,
  };

setTodos(prev ⇒ [...prev, newTodo]);
resetForm();
```

```
},
```

❓ Пояснение

- UUID гарантирует уникальность
- Состояние обновляется иммутабельно
- Форма очищается после отправки

🔷 ЭТАП 9. Отображение списка задач

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Код

```
<ul>
  {todos.map(todo ⇒ (
<li key={todo.id}>
<strong
style={{
textDecoration:todo.completed ? 'line-through': 'none',
      }}
     >
      {todo.title}
</strong>

<p>{todo.description}</p>

<button onClick={() ⇒ toggleTodoStatus(todo.id)}>
      {todo.completed ? 'Вернуть в работу' : 'Завершить'}
</button>
```

```
<button onClick={() ⇒ deleteTodo(todo.id)}>
      Удалить
</button>
</li>
 ))}
</ul>
```

🔷 ЭТАП 10. Изменение статуса задачи

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Функция

```
const toggleTodoStatus = (id:string) ⇒ {
setTodos(prev ⇒
   prev.map(todo ⇒
     todo.id === id
       ? { ...todo,completed: !todo.completed }
       : todo
   )
 );
};
```

❓ Пояснение

- Используется `map`
- Меняется только одна задача
- Остальные остаются без изменений
- Нет мутаций

🔷 ЭТАП 11. Удаление задачи (КЛЮЧЕВОЙ ЭТАП)

📄 Файл

`src/components/TodoForm.tsx`

🛠️ Функция удаления

```
constdeleteTodo = (id:string) ⇒ {
setTodos(prev ⇒ prev.filter(todo ⇒ todo.id !== id));
};
```

🔍 Подробный разбор

- `id` — стабильный идентификатор
- `filter` создаёт новый массив
- Удаляется **конкретная задача**
- Порядок и индексы не важны
- React корректно обновляет UI

❗ **Использование** `index` **запрещено**