🧩 Итоговое задание: Приложение **"Movies Library"** (React + Redux + TypeScript)

🎯 Цель задания

Создать приложение для управления списком фильмов с возможностью:

- добавления фильмов;
- редактирования их названий;
- удаления фильмов.

Хранение данных реализуется с помощью **чистого Redux** и **combineReducers**, а типизация — через **TypeScript**.

📂 Структура проекта

Создайте в папке `src/` следующую структуру:

```
src/
 │
 ├── Movies/
```

```
│   ├── MovieCreation.tsx
│   ├── MovieEdit.tsx
│   ├── Movies.tsx
│   └── moviesReducer.ts
│
├── types/
│   ├── Action.ts
│   ├── Movie.ts
│   └── MovieCredentials.ts
│
└── store/
    └── index.ts
```

🪄 Этап 1. Типизация данных

В папке **types** создайте три файла, чтобы описать структуру данных и типы действий:

1. `MovieCredentials.ts`

Создайте интерфейс, описывающий базовые свойства фильма:

- `title: string`
- `genre: string`
- `country: string`
- `releaseDate: string`

Этот интерфейс будет использоваться при создании новых фильмов.

---

2. `Movie.ts`

Создайте интерфейс, который **наследует** `MovieCredentials` и добавляет уникальный идентификатор:

- `id: string`

---

3. `Action.ts`

Определите тип действия (**Action type**) для трёх операций:

- `movies/add` — добавление фильма;
- `movies/delete` — удаление фильма по `id`;
- `movies/editTitle` — изменение названия фильма.

Каждое действие должно включать:

- свойство `type` (строковый литерал);
- свойство `payload`, соответствующее типу данных (новый фильм, `id` или объект `{ id, newTitle }`).

⚙️ Этап 2. Реализация редьюсера

Создайте файл **Movies/moviesReducer.ts**.

1. Импортируйте интерфейсы `Movie` и `Action`.
2. Определите начальное состояние — массив фильмов, содержащий хотя бы один фильм (для примера).
3. Реализуйте редьюсер `moviesReducer(state, action)`, который:
   - При типе `'movies/add'` добавляет новый фильм с уникальным `id`.
   - При типе `'movies/delete'` удаляет фильм по `id`.
   - При типе `'movies/editTitle'` изменяет `title` у фильма с указанным `id`.

Для генерации `id` можно использовать библиотеку **uid**.

🧠 Этап 3. Настройка Redux-хранилища

Создайте файл **store/index.ts**.

1. Импортируйте функции `createStore` и `combineReducers` из `redux`.
2. Импортируйте `moviesReducer`.
3. Используйте `combineReducers` для создания корневого редьюсера:

```
const rootReducer = combineReducers({
   movies: moviesReducer
});
```

4. Создайте store:

```
const store = createStore(rootReducer);
```

5. Экспортируйте:
   - `store`;
   - тип состояния `RootState`, который можно получить как `ReturnType<typeof rootReducer>`.
6. Оберните приложение в `<Provider store={store}>`, чтобы передать Redux-хранилище всему приложению.

🧩 Этап 4. Компонент создания фильма — MovieCreation.tsx

Создайте форму для добавления нового фильма.

Компонент должен включать:

1. **Локальное состояние** (через `useState`) для хранения значений полей:
   - `title`, `genre`, `country`, `releaseDate`, `error`.
2. **Валидацию полей**:
   - Проверяйте, что все поля заполнены.
   - Если нет — выводите сообщение об ошибке красным цветом.
3. **Отправку данных**:
   - При нажатии кнопки «Создать» проверяйте поля.

- Если всё корректно — вызывайте `dispatch` с действием `movies/add` и `payload` из введённых данных.
- После отправки очищайте форму и ошибки.

🧾 Этап 5. Компонент списка фильмов — Movies.tsx

Компонент отвечает за отображение всех фильмов из Redux-хранилища.

1. С помощью **useSelector** получите массив фильмов (`state.movies`).
2. Для каждого фильма выведите:
   - Название.
   - Страну, жанр, дату выхода.
   - Кнопку удаления.
   - Компонент редактирования (`<MovieEdit />`).
3. При нажатии на иконку удаления вызывайте `dispatch({ type: 'movies/delete', payload: id })`.

✏️ Этап 6. Компонент редактирования фильма — MovieEdit.tsx

Реализуйте функционал редактирования названия фильма:

1. При клике на иконку ✏️ появляется форма ввода нового названия.
2. После отправки формы:
   - вызывается `dispatch` с типом `'movies/editTitle'` и `payload: { id, newTitle }`;
   - форма скрывается, поле очищается.
3. Для отображения иконки используйте **Material UI** (`EditIcon`).

🧱 Этап 7. Комбинация всех компонентов

На странице (например, `App.tsx`):

1. Подключите Redux через `<Provider>`.
2. Отобразите компоненты:

```
<MovieCreation />
<Movies />
```

3. Убедитесь, что все изменения (добавление, редактирование, удаление) сразу видны без перезагрузки страницы.

🌟 Этап 8. Визуальное оформление (по желанию)

- Используйте иконки:
  - ✏️ `EditIcon` — редактирование;
  - 🗑️ `DeleteIcon` — удаление.
- Можно добавить простое оформление через inline-стили или CSS-модули.

✅ Проверка результата

Приложение должно позволять:

Действие	Описание
➕ Добавить фильм	Через форму с валидацией
🗑️ Удалить фильм	Через иконку в списке
✏️ Изменить название	Через форму редактирования
🔄 Отображение изменений	Без перезагрузки страницы