МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные системы»

Дисциплина «Интернет-программирование»

**Лабораторная работа № 5**

Разработка компонентов на React

Выполнил:

студент гр. ПИбд-23

Пазушкин И.П.

Проверил:

преподаватель

Истюков Т.А..

Ульяновск,   
2025 г.

**Задание лабораторной работы**

1. Необходимо сформировать окружение проекта для разработки SPA на React (см. лекцию 5 и [пример](https://lms.ulstu.ru/mod/resource/view.php?id=65677) 5).
2. На основе логики предыдущих работ необходимо разработать компоненты с использованием React для реализации операций добавления, редактирования и удаления данных о некоторой сущности предметной области (см. [пример](https://lms.ulstu.ru/mod/resource/view.php?id=65677) 5). Необходимо разработать компонент-форму для добавления и редактирования сведений о сущностях проблемной области на некоторую страницу (карточки с товарами, карточки фильмов, карточки друзей и т. д). Также необходимо разработать компоненты для отображения сведений об объекте. При добавлении записи динамически должны быть созданы и добавлены в DOM необходимые компоненты. При редактировании или удалении записи должно быть обновлено содержимое отдельных компонентов или отдельные компоненты должны быть удалены из DOM.
3. Макет страницы и компонентов может быть простым и не использовать большое количество стилей. Результатом должны быть адекватно реализованные react-компоненты с чистыми функциями рендеринга, обработчиками событий и состояниями. Обработчики событий и состояния могут быть содержаться не во всех компонентах, но должны быть реализованы.
4. Файлы проекта и отчет должны быть загружены в git-репозиторий на сервере <http://student.git.athene.tech/>

**Порядок выполнения работы**

Структура и окружение проекта было реализовано мной не по примеру преподавателя. Мои компоненты для реализации MVC шаблона панели администратора на React:

* **AdminPage** – основной компонент, содержащий в себе все остальные.
* **ListView** – компонент, содержащий список предметов конкретной категории моей предметной области.
* **AdminForm** – компонент, определяющий другой компонент, в зависимости от характеристик.
* **BookForm** – компонент, форма для книг.
* **CertificateForm** – компонент, форма для подарочных сертификатов.

Реализация CRUD-операций:

* Добавление. Через форму данные отправляются на сервер, где сохраняются в json формате.
* Редактирование. При выборе элемента в списке открывается форма для его редактирования, и при повторной отправке его данные меняются.
* Удаление. В списке элементов есть специальная кнопка для удаления элемента.

**Скриншоты**

1. Добавление:

**Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.**

1. Редактирование:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

1. Удаление:

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Исходный код**

Файл AdminPage:

import React from 'react';

import { ListView } from '../features/ListView/ListView';

import { useElibStore, type Store } from '../../store/store';

import { z } from 'zod';

import { AdminForm } from '../features/AdminForm/AdminForm';

import type { Book, Certificate } from '../../types';

import { Button } from '../ui/Button';

*const* optionVariants = z.object({

  value: z.enum(['books', 'certificates']),

});

*type* OptionVariants = z.infer<typeof optionVariants>['value'];

export *const* AdminPage = (): React.JSX.Element *=>* {

*const* [currentList, setCurrentList] = React.useState<OptionVariants>(

    () *=>* 'books',

  );

*const* [selectedItem, setSelectedItem] = React.useState<Book | Certificate>();

*const* [loading, setLoading] = React.useState<*boolean*>(() *=>* true);

*const* books = useElibStore((*state*: Store) *=>* *state*.books);

*const* certificates = useElibStore((*state*: Store) *=>* *state*.certificates);

*const* getBoth = useElibStore((*state*: Store) *=>* *state*.getBoth);

*const* deleteBook = useElibStore((*state*: Store) *=>* *state*.removeBook);

*const* deleteCertificate = useElibStore(

    (*state*: Store) *=>* *state*.removeCertificate,

  );

  React.useEffect(() *=>* {

    console.log('AdminPage: Loading data');

    setLoading(true);

*const* loadData = async () *=>* {

      try {

        await getBoth();

      } catch (error) {

        console.error('AdminPage: Error loading data', error);

      } finally {

        setLoading(false);

      }

    };

    loadData();

  }, [getBoth]);

*const* handleListChange = (*e*: React.ChangeEvent<HTMLSelectElement>) *=>* {

    if (optionVariants.safeParse({ value: *e*.target.value }).success) {

      setCurrentList(*e*.target.value as OptionVariants);

    }

  };

*const* handleCreateItem = () *=>* {

    console.log('click create');

    setSelectedItem(undefined);

  };

*const* onItemClick = (*id*: *number*) *=>* {

    console.log('click item', *id*);

    switch (currentList) {

      case 'books': {

*const* book = books.find((*b*: Book) *=>* *b*.id === *id*);

        setSelectedItem(book);

        break;

      }

      case 'certificates': {

*const* certificate = certificates.find((*c*: Certificate) *=>* *c*.id === *id*);

        setSelectedItem(certificate);

        break;

      }

      default:

        break;

    }

  };

*const* handleDeleteItem = (*id*: *number*) *=>* {

    console.log('click delete', *id*);

    switch (currentList) {

      case 'books': {

        deleteBook(*id*);

        break;

      }

      case 'certificates': {

        deleteCertificate(*id*);

        break;

      }

      default:

        break;

    }

  };

  return (

    <main className="flex-1 flex flex-col">

      {loading ? (

        <div className="flex items-center justify-center w-full h-full flex-1">

          <p className="text-xl">Загрузка данных...</p>

        </div>

      ) : (

        <div className="flex flex-1">

          <section className="border-r px-1 py-3 w-3/12">

            <select

              onChange={handleListChange}

              className="border flex flex-column w-full transition p-2 mb-4"

              value={currentList}

            >

              <option value="books">Книги ({books?.length || 0})</option>

              <option value="certificates">

                Сертификаты ({certificates?.length || 0})

              </option>

            </select>

            <*ListView*

              items={currentList === 'books' ? books : certificates}

              onItemClick={onItemClick}

              selectedId={selectedItem?.id}

              handleDeleteItem={handleDeleteItem}

            />

            <*Button* className="px-4 py-3" onClick={handleCreateItem}>

              Создать

            </*Button*>

          </section>

          <section className="flex-1 flex justify-center items-center">

            <*AdminForm* currentList={currentList} initialValue={selectedItem} />

          </section>

        </div>

      )}

    </main>

  );

};

export default AdminPage;

Файл ListView

import React from 'react';

import { cn } from '../../../lib/cn';

*type* Props = {

  items: { id: *number*; name: *string* }[];

  selectedId?: *number*;

  onItemClick: (*id*: *number*) *=>* *void*;

  handleDeleteItem: (*id*: *number*) *=>* *void*;

};

export *const* ListView = ({

*items*,

*selectedId*,

  onItemClick,

  handleDeleteItem,

}: Props): React.JSX.Element *=>* {

*const* handleItemClick = (*id*: *number*) *=>* {

    onItemClick(*id*);

  };

  return (

    <>

      <ul className="flex flex-col gap-2 p-1">

        {*items*?.map(*item* *=>* (

          <li

            className={cn(

              'w-full flex justify-between text-justify overflow-ellipsis border border-main p-2 rounded hover:bg-gray hover:border hover:border-dark',

              {

                'bg-gray': *selectedId* && *selectedId* === *item*.id,

              },

            )}

            key={*item*.id}

            onClick={() *=>* handleItemClick(*item*.id)}

          >

            {*item*.name}

            <button

              onClick={() *=>* handleDeleteItem(*item*.id)}

              className="text-red-500 hover:text-red-700 "

            >

              x

            </button>

          </li>

        ))}

      </ul>

    </>

  );

};

Файл AdminForm

import React from 'react';

import { BookForm } from '../BookForm/BookForm';

import type { Certificate, Book, List } from '../../../types';

import { CertificateForm } from '../CertificateForm/CertificateForm';

*type* Props = {

  initialValue?: Book | Certificate;

  currentList: List;

};

export *const* AdminForm = ({

*initialValue*,

*currentList*,

}: Props): React.JSX.Element *=>* {

  console.log(*initialValue*);

  return (

    <>

      {*currentList* == 'books' && <*BookForm* initialValue={*initialValue*} />}

      {*currentList* == 'certificates' && (

        <*CertificateForm* initialValue={*initialValue*} />

      )}

      {*currentList* !== 'books' && *currentList* !== 'certificates' && (

        <p>не знаю что это такое</p>

      )}

    </>

  );

};

Файл BookForm

import React from 'react';

import { useForm } from 'react-hook-form';

import { useElibStore } from '../../../store/store';

import { zodResolver } from '@hookform/resolvers/zod';

import {

  FormDataEdit,

  formDataSchemaEdit,

  FormDataAdd,

  formDataSchemaAdd,

} from './formDataSchema';

import type { Book } from '../../../types';

import { Button } from '../../ui/Button';

import {

  AuthorsCheckboxesEdit,

  AuthorsCheckboxesAdd,

} from './AuthorCheckboxes';

import { GenresCheckboxesEdit, GenresCheckboxesAdd } from './GenreCheckboxes';

*type* Props = {

  initialValue?: Book;

};

export *const* BookForm = ({ *initialValue* }: Props): React.JSX.Element *=>* {

*const* isEditing = !!*initialValue*;

*const* {

    control: controlEdit,

    register: registerEdit,

    reset: resetEdit,

    formState: { errors: errorsEdit },

    handleSubmit: handleSubmitEdit,

  } = useForm<FormDataEdit>({

    resolver: zodResolver(formDataSchemaEdit),

  });

*const* {

    control: controlAdd,

    register: registerAdd,

    reset: resetAdd,

    formState: { errors: errorsAdd },

    handleSubmit: handleSubmitAdd,

  } = useForm<FormDataAdd>({

    resolver: zodResolver(formDataSchemaAdd),

  });

*const* register = isEditing ? registerEdit : registerAdd;

*const* errors = isEditing ? errorsEdit : errorsAdd;

*const* reset = isEditing ? resetEdit : resetAdd;

*const* authors = useElibStore(*state* *=>* *state*.authors);

*const* genres = useElibStore(*state* *=>* *state*.genres);

*const* updateBook = useElibStore(*state* *=>* *state*.updateBook);

*const* createBook = useElibStore(*state* *=>* *state*.addBook);

  React.useEffect(() *=>* {

    if (*initialValue*) {

*const* authorIds = *Array*.isArray(*initialValue*.authors)

        ? *initialValue*.authors.map(*author* *=>* *author*.id)

        : [];

*const* genreIds = *Array*.isArray(*initialValue*.genres)

        ? *initialValue*.genres.map(*genre* *=>* *genre*.id)

        : [];

      console.log('Извлеченные ID авторов:', authorIds);

      console.log('Извлеченные ID жанров:', genreIds);

      resetEdit({

        id: *initialValue*.id.toString(),

        name: *initialValue*.name,

        authorIds,

        genreIds,

        cover: undefined,

        file: undefined,

      });

    } else {

      resetAdd({

        id: *Date*.now().toString(),

        name: '',

        authorIds: [],

        genreIds: [],

        cover: undefined,

        file: undefined,

      });

    }

  }, [*initialValue*, resetEdit, resetAdd]);

*const* processFormData = (*data*: FormDataEdit | FormDataAdd) *=>* {

    console.log('Данные формы перед отправкой:', *data*);

*const* formData = new *FormData*();

    formData.append('id', *data*.id || *Date*.now().toString());

    if (*data*.name) formData.append('name', *data*.name);

    if (*data*.authorIds && *data*.authorIds.length > 0) {

*const* authorObjects = *data*.authorIds.map(*authorId* *=>* {

*const* author = authors.find(*a* *=>* *a*.id === *authorId*);

        return {

          id: *authorId*,

          name: author ? author.name : 'Неизвестный автор',

        };

      });

      formData.append('authors', JSON.stringify(authorObjects));

    } else {

      formData.append('authors', JSON.stringify([]));

    }

    if (*data*.genreIds && *data*.genreIds.length > 0) {

*const* genreObjects = *data*.genreIds.map(*genreId* *=>* {

*const* genre = genres.find(*g* *=>* *g*.id === *genreId*);

        return {

          id: *genreId*,

          name: genre ? genre.name : 'Неизвестный жанр',

        };

      });

      formData.append('genres', JSON.stringify(genreObjects));

    } else {

      formData.append('genres', JSON.stringify([]));

    }

    if (*data*.cover) formData.append('poster', *data*.cover);

    if (*data*.file) formData.append('file', *data*.file);

    console.log(

      'FormData для отправки:',

*Object*.fromEntries(formData.entries()),

    );

    return formData;

  };

*const* handleSubmitEditForm = async (*data*: FormDataEdit) *=>* {

    try {

*const* formData = processFormData(*data*);

      await updateBook(+*data*.id, formData);

      console.log('Книга успешно обновлена');

      refreshBooksList();

      reset();

    } catch (error) {

      console.error('Ошибка при обновлении книги:', error);

      alert(error);

    }

  };

*const* handleSubmitAddForm = async (*data*: FormDataAdd) *=>* {

    try {

*const* formData = processFormData(*data*);

*const* newBook = await createBook(formData);

      console.log('Книга успешно создана:', newBook);

      refreshBooksList();

      reset();

    } catch (error) {

      console.error('Ошибка при создании книги:', error);

      alert(error);

    }

  };

*const* refreshBooksList = async () *=>* {

*const* fetchBooks = useElibStore.getState().fetchBooks;

    if (typeof fetchBooks === 'function') {

      try {

        await fetchBooks();

        console.log('Список книг обновлен');

      } catch (refreshError) {

        console.error('Ошибка при обновлении списка книг:', refreshError);

      }

    }

  };

  return (

    <form

      onSubmit={

        isEditing

          ? handleSubmitEdit(handleSubmitEditForm)

          : handleSubmitAdd(handleSubmitAddForm)

      }

      className="flex flex-col gap-4 border rounded p-4 w-[700px] mx-auto shadow-lg space-y-6"

    >

      <input type="hidden" {...register('id')} />

      <div>

        <label className="block text-sm font-medium text-gray-700">

          Название:

        </label>

        <input

          {...register('name')}

          placeholder="Введите название"

          className="mt-1 block w-full p-2 border border-gray-300 rounded-md focus:ring-blue-500 focus:border-blue-500"

          required={!isEditing}

        />

        {errors.name && (

          <span className="text-red-500 text-sm mt-1 block">

            {errors.name.message}

          </span>

        )}

      </div>

      <div>

        <label className="block text-sm font-medium text-gray-700">

          Авторы:

        </label>

        {isEditing ? (

          <*AuthorsCheckboxesEdit* control={controlEdit} authors={authors} />

        ) : (

          <*AuthorsCheckboxesAdd* control={controlAdd} authors={authors} />

        )}

        {errors.authorIds && (

          <span className="text-red-500 text-sm mt-1 block">

            {errors.authorIds.message}

          </span>

        )}

      </div>

      <div>

        <label className="block text-sm font-medium text-gray-700">

          Жанры:

        </label>

        {isEditing ? (

          <*GenresCheckboxesEdit* control={controlEdit} genres={genres} />

        ) : (

          <*GenresCheckboxesAdd* control={controlAdd} genres={genres} />

        )}

        {errors.genreIds && (

          <span className="text-red-500 text-sm mt-1 block">

            {errors.genreIds.message}

          </span>

        )}

      </div>

      <div>

        <label className="block text-sm font-medium text-gray-700">

          Обложка:

        </label>

        {*initialValue*?.cover && (

          <div className="text-sm text-gray-500 mb-2">

            Текущая обложка: {*initialValue*.cover.split('/').pop()}

          </div>

        )}

        <input

          type="file"

          accept="image/jpeg,image/png"

          {...register('cover')}

          className="mt-1 block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-gray file:text-gray-700 hover:file:bg-main"

          required={!isEditing}

        />

        {errors.cover && (

          <span className="text-red-500 text-sm mt-1 block">

            {errors.cover.message?.toString()}

          </span>

        )}

      </div>

      <div>

        <label className="block text-sm font-medium text-gray-700">

          Файл книги:

        </label>

        {*initialValue*?.path && (

          <div className="text-sm text-gray-500 mb-2">

            Текущий файл: {*initialValue*.path.split('/').pop()}

          </div>

        )}

        <input

          type="file"

          accept=".epub"

          {...register('file')}

          className="mt-1 block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-gray file:text-gray-700 hover:file:bg-main"

          required={!isEditing}

        />

        {errors.file && (

          <span className="text-red-500 text-sm mt-1 block">

            {errors.file.message?.toString()}

          </span>

        )}

      </div>

      <div className="mx-auto">

        <*Button* className="max-w-[200px] w-50" type="submit">

          {isEditing ? 'Обновить' : 'Добавить'}

        </*Button*>

      </div>

    </form>

  );

};

Файл CertificateForm

import React from 'react';

import { useForm } from 'react-hook-form';

import type { Certificate } from '../../../types';

import {

  formDataSchemaEdit,

  FormDataEdit,

  formDataSchemaAdd,

  FormDataAdd,

} from './formDataSchema';

import { useElibStore } from '../../../store/store';

import { zodResolver } from '@hookform/resolvers/zod';

import { Button } from '../../ui/Button';

*type* Props = {

  initialValue?: Certificate;

};

export *const* CertificateForm = ({ *initialValue* }: Props): React.JSX.Element *=>* {

*const* isEditing = !!*initialValue*;

*const* {

    register: registerEdit,

    formState: { errors: errorsEdit },

    reset: resetEdit,

    handleSubmit: handleSubmitEdit,

  } = useForm<FormDataEdit>({

    resolver: zodResolver(formDataSchemaEdit),

  });

*const* {

    register: registerAdd,

    formState: { errors: errorsAdd },

    reset: resetAdd,

    handleSubmit: handleSubmitAdd,

  } = useForm<FormDataAdd>({

    resolver: zodResolver(formDataSchemaAdd),

  });

*const* register = isEditing ? registerEdit : registerAdd;

*const* errors = isEditing ? errorsEdit : errorsAdd;

*const* reset = isEditing ? resetEdit : resetAdd;

  React.useEffect(() *=>* {

    if (*initialValue*) {

      console.log('Начальные данные сертификата:', *initialValue*);

      resetEdit({

        id: *initialValue*.id.toString(),

        name: *initialValue*.name,

        text: *initialValue*.text,

        img: undefined,

      });

    } else {

      resetAdd({

        id: *Date*.now().toString(),

        name: '',

        text: '',

        img: undefined,

      });

    }

  }, [*initialValue*, resetEdit, resetAdd]);

*const* updateCertificate = useElibStore(*store* *=>* *store*.updateCertificate);

*const* createCertificate = useElibStore(*store* *=>* *store*.addCertificate);

*const* processFormData = (*data*: FormDataEdit | FormDataAdd) *=>* {

    console.log('Данные формы перед отправкой:', *data*);

*const* formData = new *FormData*();

    formData.append('id', *data*.id || *Date*.now().toString());

    if (*data*.name) formData.append('name', *data*.name);

    if (*data*.text) formData.append('text', *data*.text);

    if (*data*.img) formData.append('img', *data*.img);

    console.log(

      'FormData для отправки:',

*Object*.fromEntries(formData.entries()),

    );

    return formData;

  };

*const* handleSubmitEditForm = async (*data*: FormDataEdit) *=>* {

    try {

*const* formData = processFormData(*data*);

      await updateCertificate(+*data*.id, formData);

      console.log('Сертификат успешно обновлен');

      refreshCertificatesList();

      reset();

    } catch (error) {

      console.error('Ошибка при сохранении сертификата:', error);

      alert(error);

    }

  };

*const* handleSubmitAddForm = async (*data*: FormDataAdd) *=>* {

    try {

*const* formData = processFormData(*data*);

*const* newCertificate = await createCertificate(formData);

      console.log('Сертификат успешно создан:', newCertificate);

      refreshCertificatesList();

      reset();

    } catch (error) {

      console.error('Ошибка при сохранении сертификата:', error);

      alert(error);

    }

  };

*const* refreshCertificatesList = async () *=>* {

*const* fetchCertificates = useElibStore.getState().fetchCertificates;

    if (typeof fetchCertificates === 'function') {

      try {

        await fetchCertificates();

        console.log('Список сертификатов обновлен');

      } catch (refreshError) {

        console.error(

          'Ошибка при обновлении списка сертификатов:',

          refreshError,

        );

      }

    }

  };

  return (

    <form

      onSubmit={

        isEditing

          ? handleSubmitEdit(handleSubmitEditForm)

          : handleSubmitAdd(handleSubmitAddForm)

      }

      className="flex flex-col gap-4 border rounded p-4 w-[700px] h-[600px] mx-auto shadow-lg space-y-6"

      autoComplete="on"

    >

      <input type="hidden" {...register('id')} />

      <div>

        <label className="flex flex-col">

          <span className="pl-2">Название:</span>

          <input

            type="text"

            {...register('name')}

            placeholder="Название"

            className="border p-2 rounded"

            required={!isEditing}

          />

          {errors.name && (

            <span className="text-red-500 text-sm mt-1 block">

              {errors.name.message}

            </span>

          )}

        </label>

      </div>

      <div>

        <label className="flex flex-col">

          <span className="pl-2">Текст:</span>

          <textarea

            {...register('text')}

            placeholder="Текст"

            className="border rounded p-2 resize-none"

            rows={4}

            required={!isEditing}

          />

          {errors.text && (

            <span className="text-red-500 text-sm mt-1 block">

              {errors.text.message}

            </span>

          )}

        </label>

      </div>

      <div>

        <label className="flex flex-col">

          <span className="pl-2">Изображение:</span>

          {*initialValue*?.img && (

            <div className="text-sm text-gray-500 mb-2">

              Текущий файл: {*initialValue*.img.split('/').pop()}

            </div>

          )}

          <input

            type="file"

            {...register('img')}

            className="mt-1 block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-gray file:text-gray-700 hover:file:bg-main"

            required={!isEditing}

          />

          {errors.img && (

            <span className="text-red-500 text-sm mt-1 block">

              {errors.img.message?.toString()}

            </span>

          )}

        </label>

      </div>

      <div className="mx-auto">

        <*Button* className="max-w-[200px] w-50" type="submit">

          {isEditing ? 'Обновить' : 'Добавить'}

        </*Button*>

      </div>

    </form>

  );

};