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 № 6**

React: разработка полноценного веб-приложения

Выполнил:

студент гр. ПИбд-23

Пазушкин И.П.

Проверил:

преподаватель

Истюков Т.А.

Ульяновск,   
2025 г.

**Задание лабораторной работы**

1. В рамках данной работы необходимо реализовать все функциональные возможности веб-приложения, которые были запланированы при проектировании: создание, редактирование и удаление карточек объектов, отображение всех необходимых элементов страницы, оформление покупок, подписок и т. д.

2. Необходимо реализовать переход по страницам сайта с помощью маршрутизации (SPA).

3. Следует выделить отдельные части страниц веб-приложения в React-компоненты.

4. Бизнес-логика веб-приложения должна быть вынесена в отдельные кастомные хуки.

5. При проектировании компонентов следует уделить особое внимание количеству и правильности использования состояний компонента (useState) и корректности использования useEffect.

6. Приложение должно использовать в серверной части пакет json-server. Все запросы к серверу должны выполняться через протокол HTTP с учетом особенностей архитектурного стиля REST.

7. Файлы проекта и отчет должны быть загружены в git-репозиторий на сервере http://student.git.athene.tech/

**Порядок выполнения работы**

Да короче, сделал не по примерам, примеры даже не смотрел, есть react-router-dom для реализации SPA и роутинга. (интересно, а читают ли отчеты?) Поставил крутой менеджер состояний Zustand, реализуя его подход слайсов, то есть разбиения одного глобального состояния на несколько более мелких. Для возможности читать книги использовал <https://github.com/gerhardsletten/react-reader>.

**Скриншоты**

*Страница MainPage.tsx:* *Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница LoginPage.tsx:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница RegisterPage.tsx:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница CertificatesPage.tsx:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница UserPage.tsx:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница BookPage.tsx:*

*Изображение выглядит как текст, снимок экрана, человек, Веб-сайт

Контент, сгенерированный ИИ, может содержать ошибки.*

*Страница AdminPage.tsx*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

**Функциональные возможности веб-приложения**

* *Реализация чтения книг:*

*Страница BookPage.tsx:*

*Изображение выглядит как текст, Человеческое лицо, компьютер, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.*

* *Реализации карточек товаров:*

*Страница MainPage.tsx:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*1. Добавление:*

*Страница AdminPage.tsx:*

*Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.*

*2. Редактирование:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*3. Удаление:*

*Изображение выглядит как снимок экрана, Прямоугольник, символ, линия

Контент, сгенерированный ИИ, может содержать ошибки.*

*Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.*

**Исходный код**

**eslint.config.js**

import js from '@eslint/js' import globals from 'globals' import reactHooks from 'eslint-plugin-react-hooks' import reactRefresh from 'eslint-plugin-react-refresh' import tseslint from 'typescript-eslint' export default tseslint.config( { ignores: ['dist'] }, { extends: [js.configs.recommended, ...tseslint.configs.recommended], files: ['\*\*/\*.{ts,tsx}'], languageOptions: { ecmaVersion: 2020, globals: globals.browser, }, plugins: { 'react-hooks': reactHooks, 'react-refresh': reactRefresh, }, rules: { ...reactHooks.configs.recommended.rules, 'react-refresh/only-export-components': [ 'warn', { allowConstantExport: true }, ], }, }, )

**index.html**

<!DOCTYPE html> <html lang="ru"> <head> <meta charset="UTF-8" /> <link rel="icon" type="image/svg+xml" href="/icon.svg" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <meta name="description" content="Elib - онлайн-библиотека для чтения книг" /> <title>Elib</title> </head> <body class="flex flex-col"> <div id="root" class="flex-1"></div> <script type="module" src="/src/main.tsx"></script> </body> </html>

**package.json**

{ "name": "elib", "private": true, "version": "0.0.0", "type": "module", "scripts": { "dev": "vite", "build": "tsc -b && vite build", "lint": "eslint .", "preview": "vite preview", "server": "bun run server/index.ts" }, "dependencies": { "@hookform/resolvers": "^5.0.1", "@tailwindcss/vite": "^4.1.3", "clsx": "^2.1.1", "cors": "^2.8.5", "express": "^4.21.2", "express-fileupload": "^1.5.1", "multer": "^1.4.5-lts.2", "react": "^19.0.0", "react-dom": "^19.0.0", "react-hook-form": "^7.55.0", "react-reader": "^2.0.13", "react-router-dom": "^7.5.0", "svelte": "^5.25.8", "tailwind-merge": "^3.2.0", "tailwindcss": "^4.1.3", "zod": "^3.24.2", "zustand": "^5.0.3" }, "devDependencies": { "@eslint/js": "^9.21.0", "@types/multer": "^1.4.12", "@sveltejs/vite-plugin-svelte": "^5.0.3", "@types/cors": "^2.8.17", "@types/node": "^22.15.14", "@types/react": "^19.0.10", "@types/react-dom": "^19.0.4", "@vitejs/plugin-react": "^4.3.4", "eslint": "^9.21.0", "eslint-plugin-react-hooks": "^5.1.0", "eslint-plugin-react-refresh": "^0.4.19", "globals": "^15.15.0", "typescript": "~5.7.2", "typescript-eslint": "^8.24.1", "vite": "^6.2.0" } }

**public\authors.json**

[ { "id": 1, "name": "Артур Конан Дойл" }, { "id": 2, "name": "Джоан Роулинг" }, { "id": 3, "name": "Джордж Мартин" }, { "id": 4, "name": "Лев Толстой" }, { "id": 5, "name": "Федор Достоевский" }, { "id": 6, "name": "Александр Пушкин" }, { "id": 7, "name": "Михаил Булгаков" }, { "id": 8, "name": "Антон Чехов" }, { "id": 9, "name": "Иван Тургенев" }, { "id": 10, "name": "Николай Гоголь" } ]

**public\books.json**

[ { "id": 1, "name": "Шерлок Холмс1", "authors": [ { "id": 0, "name": "Артур Конан Дойл" } ], "genres": [ { "id": 1, "name": "Детектив" } ], "cover": "/images/covers/sherlockholmes.jpg", "path": "sherlockholmes.epub" }, { "id": 2, "name": "Гарри Поттер и философский камень", "authors": [ { "id": 0, "name": "Артур Конан Дойл" } ], "genres": [ { "id": 1, "name": "Детектив" } ], "cover": "/images/covers/harrypotter.jpg", "path": "harrypotter.epub" } ]

**public\certificates.json**

[ { "id": 1, "name": "Подписка на 3 месяца", "text": "Всё, от Гарри Поттера до Незнайки. Открывайте для себя новые удивительные миры и занимательные истории!", "img": "/images/posters/book-shkaf.svg" } ]

**public\genres.json**

[ { "id": 1, "name": "Детектив" }, { "id": 2, "name": "Фэнтези" }, { "id": 3, "name": "Научная фантастика" }, { "id": 4, "name": "Приключения" }, { "id": 5, "name": "Ужасы" }, { "id": 6, "name": "Триллер" }, { "id": 7, "name": "Романтика" }, { "id": 8, "name": "Комедия" }, { "id": 9, "name": "Драма" }, { "id": 10, "name": "Исторический роман" }, { "id": 11, "name": "Фантастика" }, { "id": 12, "name": "Мистика" } ]

**server\constants.ts**

import { fileURLToPath } from 'node:url'; import path from 'node:path'; const \_\_filename = fileURLToPath(import.meta.url); const \_\_dirname = path.dirname(\_\_filename); export const BOOKS\_PATH = path.join(\_\_dirname, '..', 'public', 'books.json'); export const CERTIFICATES\_PATH = path.join( \_\_dirname, '..', 'public', 'certificates.json', ); export const GENRES\_PATH = path.join(\_\_dirname, '..', 'public', 'genres.json'); export const AUTHORS\_PATH = path.join( \_\_dirname, '..', 'public', 'authors.json', ); export interface Book { id: number; name: string; authors: Author[]; genres: Genre[]; cover: string; path: string; } export interface Certificate { id: number; name: string; text: string; img: string; } export interface Genre { id: number; name: string; } export interface Author { id: number; name: string; }

**server\index.ts**

import express, { type Request, type RequestHandler } from 'express'; import cors from 'cors'; import fileUpload from 'express-fileupload'; import path from 'path'; import { CERTIFICATES\_PATH, Certificate, BOOKS\_PATH, Book, GENRES\_PATH, Genre, AUTHORS\_PATH, Author, } from './constants'; import { updateItemJson, updateJson, deleteItem, checkItem, readJson, } from './IO'; const app = express(); const port = 3000; app.use(cors()); app.use(express.json()); app.use(fileUpload()); app.use('/public', express.static(path.join(\_\_dirname, '..', 'public'))); // Books app.get('/books', async (req, res) => { try { const books = (await readJson(BOOKS\_PATH)) as Book[]; const authors = await readJson(AUTHORS\_PATH); const genres = await readJson(GENRES\_PATH); const booksWithFullData = books.map(book => ({ ...book, authors: book.authors .map(({ id: authorId }) => authors.find((a: Author) => a.id === authorId), ) .filter(Boolean), genres: book.genres .map(({ id: genreId }) => genres.find((g: Genre) => g.id === genreId)) .filter(Boolean), })); res.json(booksWithFullData); } catch (e) { res.status(500).json({ message: 'Ошибка при получении книг' }); } }); app.post('/books', async (req, res) => { if (!req.files || !req.files.file || !req.files.poster) { return res.status(400).json({ message: 'Файлы не загружены' }); } const file = req.files.file as fileUpload.UploadedFile; const poster = req.files.poster as fileUpload.UploadedFile; if (!file.name.endsWith('.epub')) { return res.status(400).json({ message: 'Загрузите файл в формате EPUB' }); } if (!poster.name.endsWith('.jpg') && !poster.name.endsWith('.png')) { return res .status(400) .json({ message: 'Загрузите обложку в формате JPG или PNG' }); } let authors = []; let genres = []; // Пытаемся распарсить JSON-строки с авторами и жанрами if (req.body.authors) { try { authors = JSON.parse(req.body.authors); console.log('Parsed authors:', authors); } catch (e) { console.error('Error parsing authors:', e); } } if (req.body.genres) { try { genres = JSON.parse(req.body.genres); console.log('Parsed genres:', genres); } catch (e) { console.error('Error parsing genres:', e); } } const bookData: Book = { id: +(Math.random() \* 1000).toFixed(0), // Will be set by updateJson cover: `/images/covers/${poster.name}`, authors, genres, name: req.body.name, path: `/books/${file.name}`, }; if (await checkItem(bookData, BOOKS\_PATH)) { return res .status(400) .json({ message: 'Книга с таким названием уже существует' }); } try { await file.mv(path.join(\_\_dirname, '..', 'public', file.name)); await poster.mv( path.join(\_\_dirname, '..', 'public', 'images', 'covers', poster.name), ); const newBook = await updateJson(bookData, BOOKS\_PATH); res.status(200).json(newBook); } catch (err) { console.error(err); res.status(500).json({ message: 'Ошибка при сохранении книги' }); } }); interface FileUploadRequest extends Request { files?: { [key: string]: fileUpload.UploadedFile | fileUpload.UploadedFile[]; }; } const putBookHandler: RequestHandler = async (req, res) => { try { const bookId = parseInt(req.params.id); if (isNaN(bookId)) { res.status(400).json({ error: 'Invalid book ID' }); return; } const existingBooks = (await readJson(BOOKS\_PATH)) as Book[]; const existingBook = existingBooks.find(b => b.id === bookId); if (!existingBook) { res.status(404).json({ error: 'Book not found' }); return; } const typedReq = req as FileUploadRequest; const coverFile = typedReq.files?.cover as | fileUpload.UploadedFile | undefined; let authors = existingBook.authors; let genres = existingBook.genres; if (req.body.authors) { try { authors = JSON.parse(req.body.authors); console.log('Parsed authors:', authors); } catch (e) { console.error('Error parsing authors:', e); } } if (req.body.genres) { try { genres = JSON.parse(req.body.genres); console.log('Parsed genres:', genres); } catch (e) { console.error('Error parsing genres:', e); } } const updatedBook: Book = { id: bookId, name: req.body.name || existingBook.name, authors, genres, cover: coverFile ? `/images/covers/${coverFile.name}` : existingBook.cover, path: existingBook.path, }; if (coverFile) { await coverFile.mv( path.join( \_\_dirname, '..', 'public', 'images', 'covers', coverFile.name, ), ); } const result = await updateItemJson(updatedBook, BOOKS\_PATH); res.json(result); } catch (error) { console.error('Error updating book:', error); res.status(500).json({ error: 'Failed to update book' }); } }; app.put('/books/:id', putBookHandler); app.delete('/books/:id', async (req, res) => { try { await deleteItem(req.params.id, BOOKS\_PATH); res.status(200).json({ message: 'Книга удалена' }); } catch (e) { res.status(500).json({ message: 'Ошибка при удалении книги' }); } }); // Certificates app.get('/certificates', async (req, res) => { try { const certificates = await readJson(CERTIFICATES\_PATH); res.json(certificates); } catch (e) { res.status(500).json({ message: 'Ошибка при получении сертификатов' }); } }); app.post('/certificates', async (req, res) => { if (!req.files || !req.files.img) { return res.status(400).json({ message: 'Изображение не загружено' }); } const img = req.files.img as fileUpload.UploadedFile; if (!img.name.endsWith('.jpg') && !img.name.endsWith('.png')) { return res .status(400) .json({ message: 'Загрузите изображение в формате JPG или PNG' }); } const certData: Certificate = { id: 0, // Will be set by updateJson name: req.body.name, text: req.body.text, img: `/images/posters/${img.name}`, }; if (await checkItem(certData, CERTIFICATES\_PATH)) { return res .status(400) .json({ message: 'Сертификат с таким названием уже существует' }); } try { await img.mv( path.join(\_\_dirname, '..', 'public', 'images', 'posters', img.name), ); const newCert = await updateJson(certData, CERTIFICATES\_PATH); res.status(200).json(newCert); } catch (err) { console.error(err); res.status(500).json({ message: 'Ошибка при добавлении сертификата' }); } }); const getCertificateHandler: RequestHandler = async (req, res) => { try { const certificateId = parseInt(req.params.id); if (isNaN(certificateId)) { res.status(400).json({ error: 'Invalid certificate ID' }); return; } const certificates = (await readJson(CERTIFICATES\_PATH)) as Certificate[]; const certificate = certificates.find( (c: Certificate) => c.id === certificateId, ); if (!certificate) { res.status(404).json({ error: 'Certificate not found' }); return; } res.json({ id: certificate.id, text: certificate.text, img: certificate.img, }); } catch (error) { console.error('Error getting certificate:', error); res.status(500).json({ error: 'Failed to get certificate' }); } }; app.get('/certificates/:id', getCertificateHandler); app.put('/certificates/:id', async (req, res) => { try { const certId = parseInt(req.params.id); if (isNaN(certId)) { return res.status(400).json({ message: 'Некорректный ID сертификата' }); } const existingCerts = await readJson(CERTIFICATES\_PATH); const existingCert = existingCerts.find( (c: Certificate) => c.id === certId, ); if (!existingCert) { return res.status(404).json({ message: 'Сертификат не найден' }); } const img = req.files?.img as fileUpload.UploadedFile | undefined; const updatedCert: Certificate = { ...existingCert, name: req.body.name || existingCert.name, text: req.body.text || existingCert.text, img: img ? `/images/posters/${img.name}` : existingCert.img, id: certId, }; if (img) { await img.mv( path.join( \_\_dirname, '..', 'public', 'images', 'certificates', img.name, ), ); } const result = await updateItemJson(updatedCert, CERTIFICATES\_PATH); res.status(200).json(result); } catch (e) { console.error('Ошибка при обновлении сертификата:', e); res.status(500).json({ message: 'Ошибка при обновлении сертификата' }); } }); app.delete('/certificates/:id', async (req, res) => { try { await deleteItem(req.params.id, CERTIFICATES\_PATH); res.status(200).json({ message: 'Сертификат удален' }); } catch (e) { res.status(500).json({ message: 'Ошибка при удалении сертификата' }); } }); // Genres app.get('/genres', async (req, res) => { try { const genres = await readJson(GENRES\_PATH); res.json(genres); } catch (e) { res.status(500).json({ message: 'Ошибка при получении жанров' }); } }); // Authors app.get('/authors', async (req, res) => { try { const authors = await readJson(AUTHORS\_PATH); res.json(authors); } catch (e) { res.status(500).json({ message: 'Ошибка при получении авторов' }); } }); app.listen(port, () => { console.log(`Server is running on port ${port}`); });

**server\IO.ts**

import fs from 'fs/promises'; import { Book, Certificate, Author, Genre } from './constants'; function getEntityType(item: Book | Certificate): string { if ('authors' in item && 'genres' in item) return 'Book'; if ('img' in item && 'text' in item) return 'Certificate'; return 'Unknown'; } export async function updateItemJson( item: Book | Certificate, path: string, ): Promise<Book | Certificate> { try { console.log( `Updating entity of type: ${getEntityType(item)} at path: ${path}`, ); const data = await readJson(path); const itemId = item.id; if (typeof itemId !== 'number' || isNaN(itemId)) { throw new Error('Некорректный ID объекта'); } const indexToUpdate = data.findIndex( (i: Book | Certificate) => i.id === itemId, ); if (indexToUpdate === -1) { throw new Error(`Элемент с ID ${itemId} не найден`); } data[indexToUpdate] = item; await fs.writeFile(path, JSON.stringify(data, null, 2), 'utf8'); return data[indexToUpdate]; } catch (error) { console.error('Ошибка при обновлении файла:', error); throw error; } } export async function updateJson( item: Book | Certificate, path: string, ): Promise<Book | Certificate> { try { console.log( `Adding entity of type: ${getEntityType(item)} at path: ${path}`, ); const data = await readJson(path); const newItem = { id: data.length ? Math.max(...data.map((i: Book | Certificate) => i.id)) + 1 : Math.random() \* 10, ...item, }; data.push(newItem); await fs.writeFile(path, JSON.stringify(data, null, 2), 'utf8'); return newItem; } catch (error) { console.error('Ошибка при обновлении файла:', error); throw error; } } export async function readJson( path: string, ): Promise<(Book | Certificate | Author | Genre)[]> { try { const data = await fs.readFile(path, 'utf8'); return JSON.parse(data); } catch (error) { console.error('Ошибка при чтении файла:', error); return []; } } export async function checkItem( item: Book | Certificate, path: string, ): Promise<boolean> { const items = await readJson(path); return items.some((i: Book | Certificate) => i.name === item.name); } export async function deleteItem(id: string, path: string): Promise<void> { try { const data = await readJson(path); const filteredData = data.filter( (item: Book | Certificate) => item.id !== parseInt(id), ); await fs.writeFile(path, JSON.stringify(filteredData, null, 2), 'utf8'); } catch (error) { console.error('Ошибка при удалении элемента:', error); throw error; } }

**src\api\api.ts**

import { getAllItems, createItem, updateItem, deleteItem } from './client'; import type { Book, Certificate, Genre, Author } from '../types'; export const getBooks = async (): Promise<Book[]> => getAllItems('books'); export const postBook = async (data: FormData): Promise<Book> => createItem('books', data); export const updateBook = async (data: FormData): Promise<Book> => updateItem('books', data.get('id') as string, data); export const deleteBook = async (id: string): Promise<void> => deleteItem('books', id); export const getCertificates = async (): Promise<Certificate[]> => getAllItems('certificates'); export const postCertificate = async (data: FormData): Promise<Certificate> => createItem('certificates', data); export const updateCertificate = async (data: FormData): Promise<Certificate> => updateItem('certificates', data.get('id') as string, data); export const deleteCertificate = async (id: string): Promise<void> => deleteItem('certificates', id); export const getGenres = async (): Promise<Genre[]> => getAllItems('genres'); export const getAuthors = async (): Promise<Author[]> => getAllItems('authors');

**src\api\client.ts**

import { Config } from '../lib/config'; import type { Book, Certificate } from '../types'; const envConfig = new Config(); const SERVER\_URL = envConfig.loadUrl(); const API\_PATHS = ['books', 'authors', 'genres', 'certificates']; const requestCounts = { books: 0, authors: 0, genres: 0, certificates: 0, other: 0, }; const makeRequest = async ( path: string, params?: string, vars?: string, method: 'GET' | 'POST' | 'PUT' | 'DELETE' = 'GET', data: FormData | null = null, ): Promise<Book[] | Certificate[] | Book | Certificate | void> => { try { const apiPath = API\_PATHS.find(p => path.startsWith(p)) || 'other'; requestCounts[apiPath as keyof typeof requestCounts]++; const requestParams = params ? `?${params}` : ''; const pathVariables = vars ? `/${vars}` : ''; const options: RequestInit = { method }; if ((method === 'POST' || method === 'PUT') && data) { options.body = data; } const response = await fetch( `${SERVER\_URL}${path}${pathVariables}${requestParams}`, options, ); if (!response.ok) { throw new Error(`Response status: ${response.status}`); } if (method === 'DELETE') return; const json = await response.json(); console.log(`API Response for ${path}:`, json); return json; } catch (error) { console.error(`API Error for ${path}:`, error); if (error instanceof SyntaxError) { throw new Error(`There was a SyntaxError: ${error.message}`); } else if (error instanceof Error) { throw new Error(`API request failed: ${error.message}`); } else { throw new Error(`Unknown API error`); } } }; export const getAllItems = (path: string, params?: string) => makeRequest(path, params) as Promise<Book[] | Certificate[]>; export const createItem = (path: string, data: FormData) => makeRequest(path, undefined, undefined, 'POST', data) as Promise< Book | Certificate >; export const updateItem = (path: string, id: string, data: FormData) => makeRequest(path, undefined, id, 'PUT', data) as Promise<Book | Certificate>; export const deleteItem = (path: string, id: string) => makeRequest(path, undefined, id, 'DELETE') as Promise<void>;

**src\App.tsx**

import { Outlet } from 'react-router-dom'; import { Header } from './components/layout/Header'; import { Footer } from './components/layout/Footer'; import { Suspense } from 'react'; function App() { return ( <> <Header /> <Suspense fallback={<div>Loading...</div>}> <Outlet /> </Suspense> <Footer /> </> ); } export { App };

**src\components\features\AdminForm\AdminForm.tsx**

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; handleEditAndAdd: () => void; }; export const AdminForm = ({ initialValue, currentList, handleEditAndAdd, }: Props): React.JSX.Element => { console.log(initialValue); return ( <> {currentList == 'books' && ( <BookForm handleEditAndAdd={handleEditAndAdd} initialValue={initialValue} /> )} {currentList == 'certificates' && ( <CertificateForm initialValue={initialValue} /> )} {currentList !== 'books' && currentList !== 'certificates' && ( <p>не знаю что это такое</p> )} </> ); };

**src\components\features\BookForm\AuthorCheckboxes.tsx**

import { Controller, Control } from 'react-hook-form'; import { Author } from '../../../types'; import { FormDataEdit, FormDataAdd } from './formDataSchema'; export const AuthorsCheckboxesEdit = ({ control, authors, }: { control: Control<FormDataEdit>; authors: Author[]; }) => ( <Controller name="authorIds" control={control} defaultValue={[]} render={({ field }) => ( <div className="mt-2 space-y-2 h-32 overflow-y-scroll"> {authors?.map(author => ( <label key={author.id} className="flex items-center space-x-2 text-sm text-gray-600" > <input type="checkbox" value={author.id} checked={field.value?.includes(author.id) || false} onChange={e => { const newValue = e.target.checked ? [...(field.value || []), author.id] : (field.value || []).filter(id => id !== author.id); console.log('Выбранные авторы:', newValue); field.onChange(newValue); }} className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded" /> <span>{author.name}</span> </label> ))} </div> )} /> ); // Компонент для отображения флажков авторов в режиме добавления export const AuthorsCheckboxesAdd = ({ control, authors, }: { control: Control<FormDataAdd>; authors: Author[]; }) => ( <Controller name="authorIds" control={control} defaultValue={[]} render={({ field }) => ( <div className="mt-2 space-y-2 h-32 overflow-y-scroll"> {authors?.map(author => ( <label key={author.id} className="flex items-center space-x-2 text-sm text-gray-600" > <input type="checkbox" value={author.id} checked={field.value?.includes(author.id) || false} onChange={e => { const newValue = e.target.checked ? [...(field.value || []), author.id] : (field.value || []).filter(id => id !== author.id); console.log('Выбранные авторы:', newValue); field.onChange(newValue); }} className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded" /> <span>{author.name}</span> </label> ))} </div> )} /> );

**src\components\features\BookForm\BookForm.tsx**

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; handleEditAndAdd: () => void; }; export const BookForm = ({ initialValue, handleEditAndAdd, }: Props): React.JSX.Element => { const isEditing = !!initialValue; console.log('rerender BookForm', 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(() => { console.log('rerender, initial: ', initialValue); 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({ id: Date.now().toString(), name: '', authorIds: [], genreIds: [], cover: undefined, file: undefined, }); handleEditAndAdd(); } catch (error) { console.error('Ошибка при обновлении книги:', error); alert(error); } }; const handleSubmitAddForm = async (data: FormDataAdd) => { try { const formData = processFormData(data); await createBook(formData); // refreshBooksList(); reset({ id: Date.now().toString(), name: '', authorIds: [], genreIds: [], cover: undefined, file: undefined, }); handleEditAndAdd(); } 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> ); };

**src\components\features\BookForm\formDataSchema.ts**

import { z } from 'zod'; export const formDataSchemaAdd = z.object({ id: z.string().optional(), name: z.string().min(1, { message: 'Название обязательно' }), authorIds: z .array(z.number()) .min(1, { message: 'Выберите хотя бы одного автора' }), genreIds: z .array(z.number()) .min(1, { message: 'Выберите хотя бы один жанр' }), cover: z .any() .refine( (fileList: FileList | undefined) => fileList && fileList.length > 0, { message: 'Обложка обязательна', }, ) .refine( (fileList: FileList | undefined) => !fileList || ['image/jpeg', 'image/png'].includes(fileList[0]?.type), { message: 'Обложка должна быть в формате JPG или PNG' }, ) .transform(fileList => (fileList ? fileList[0] : undefined)), file: z .any() .refine( (fileList: FileList | undefined) => fileList && fileList.length > 0, { message: 'Файл книги обязателен', }, ) .refine( (fileList: FileList | undefined) => !fileList || fileList[0]?.name.endsWith('.epub'), { message: 'Файл книги должен быть в формате EPUB' }, ) .transform(fileList => (fileList ? fileList[0] : undefined)), }); // Схема для редактирования (id обязателен, остальные поля необязательны) export const formDataSchemaEdit = z.object({ id: z.string({ required\_error: 'ID обязателен для редактирования' }), name: z.string().optional(), authorIds: z.array(z.number()).optional(), genreIds: z.array(z.number()).optional(), cover: z .any() .optional() .refine( (fileList: FileList | undefined) => !fileList || !fileList.length || ['image/jpeg', 'image/png'].includes(fileList[0]?.type), { message: 'Обложка должна быть в формате JPG или PNG' }, ) .transform(fileList => fileList && fileList.length > 0 ? fileList[0] : undefined, ), file: z .any() .optional() .refine( (fileList: FileList | undefined) => !fileList || !fileList.length || fileList[0]?.name.endsWith('.epub'), { message: 'Файл книги должен быть в формате EPUB' }, ) .transform(fileList => fileList && fileList.length > 0 ? fileList[0] : undefined, ), }); export type FormDataAdd = z.infer<typeof formDataSchemaAdd>; export type FormDataEdit = z.infer<typeof formDataSchemaEdit>;

**src\components\features\BookForm\GenreCheckboxes.tsx**

import { Controller, Control } from 'react-hook-form'; import { Genre } from '../../../types'; import { FormDataEdit, FormDataAdd } from './formDataSchema'; export const GenresCheckboxesEdit = ({ control, genres, }: { control: Control<FormDataEdit>; genres: Genre[]; }) => ( <Controller name="genreIds" control={control} defaultValue={[]} render={({ field }) => ( <div className="mt-2 space-y-2 h-32 overflow-y-scroll"> {genres?.map(genre => ( <label key={genre.id} className="flex items-center space-x-2 text-sm text-gray-600" > <input type="checkbox" value={genre.id} checked={field.value?.includes(genre.id) || false} onChange={e => { const newValue = e.target.checked ? [...(field.value || []), genre.id] : (field.value || []).filter(id => id !== genre.id); console.log('Выбранные жанры:', newValue); field.onChange(newValue); }} className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded" /> <span>{genre.name}</span> </label> ))} </div> )} /> ); // Компонент для отображения флажков жанров в режиме добавления export const GenresCheckboxesAdd = ({ control, genres, }: { control: Control<FormDataAdd>; genres: Genre[]; }) => ( <Controller name="genreIds" control={control} defaultValue={[]} render={({ field }) => ( <div className="mt-2 space-y-2 h-32 overflow-y-scroll"> {genres?.map(genre => ( <label key={genre.id} className="flex items-center space-x-2 text-sm text-gray-600" > <input type="checkbox" value={genre.id} checked={field.value?.includes(genre.id) || false} onChange={e => { const newValue = e.target.checked ? [...(field.value || []), genre.id] : (field.value || []).filter(id => id !== genre.id); console.log('Выбранные жанры:', newValue); field.onChange(newValue); }} className="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded" /> <span>{genre.name}</span> </label> ))} </div> )} /> );

**src\components\features\CertificateCard\CertificateCard.tsx**

import React from 'react'; type CertificateCardProps = Omit<Certificate, 'id'>; export const CertificateCard = ({ img, text, name, }: CertificateCardProps): React.JSX.Element => { return ( <section className="flex justify-center border rounded p-4 lg:min-h-[300px]"> <div className="flex justify-center"> <img src={img} alt={img + ' logo'} className="w-full" /> </div> <div className="flex flex-col justify-between items-center gap-2"> <h3 className="text-center">{name}</h3> <span>{text}</span> <button className="btn-primary">Купить</button> </div> </section> ); };

**src\components\features\CertificateForm\CertificateForm.tsx**

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; // Форма для редактирования const { register: registerEdit, formState: { errors: errorsEdit }, reset: resetEdit, handleSubmit: handleSubmitEdit, } = useForm<FormDataEdit>({ resolver: zodResolver(formDataSchemaEdit), }); // Форма для добавления 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); // Общий обработчик для подготовки данных формы 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; }; // Обработчик для редактирования 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); } }; // Обработчик для добавления 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> ); };

**src\components\features\CertificateForm\formDataSchema.ts**

import { z } from 'zod'; export const formDataSchemaAdd = z.object({ id: z.string().optional(), name: z.string().min(1, { message: 'Укажите название' }), text: z.string().min(1, { message: 'Укажите текст сертификата' }), img: z .any() .refine( (fileList: FileList | undefined) => fileList && fileList.length > 0, { message: 'Изображение обязательно', }, ) .refine( (fileList: FileList | undefined) => !fileList || ['image/jpeg', 'image/png', 'image/svg+xml'].includes( fileList[0]?.type, ), { message: 'Изображение должно быть в формате JPG/PNG/SVG' }, ) .transform(fileList => fileList && fileList.length > 0 ? fileList[0] : undefined, ), }); // Схема для редактирования (id обязателен, остальные поля необязательны) export const formDataSchemaEdit = z.object({ id: z.string({ required\_error: 'ID обязателен для редактирования' }), name: z.string().optional(), text: z.string().optional(), img: z .any() .optional() .refine( (fileList: FileList | undefined) => !fileList || !fileList.length || ['image/jpeg', 'image/png', 'image/svg+xml'].includes( fileList[0]?.type, ), { message: 'Изображение должно быть в формате JPG/PNG/SVG' }, ) .transform(fileList => fileList && fileList.length > 0 ? fileList[0] : undefined, ), }); export type FormDataAdd = z.infer<typeof formDataSchemaAdd>; export type FormDataEdit = z.infer<typeof formDataSchemaEdit>; // Для совместимости со старым кодом export const formDataSchema = formDataSchemaAdd; export type CertificateFormData = FormDataAdd;

**src\components\features\ListView\ListView.tsx**

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); }; console.log('rerender ListView', items); 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> </> ); };

**src\components\features\ProductCard\ProductCard.props.ts**

export interface ProductCardProps { link: string; image: string; icon: string; name: string; }

**src\components\features\ProductCard\ProductCard.tsx**

import React from 'react'; import { ProductCardProps } from './ProductCard.props'; import { cn } from '../../../lib/cn'; import { Link } from 'react-router-dom'; import { Button } from '../../ui/Button'; export const ProductCard = ({ name, image, link, icon, }: ProductCardProps): React.JSX.Element => { return ( <> <div className="flex flex-col justify-between border-2 border-gray-300 overflow-hidden shadow-lg h-[380px] w-[250px] bg-gray rounded-lg"> <div className="rounded-t-lg h-[300px] overflow-hidden bg-main flex justify-center items-center"> <img className="object-contain w-full h-full p-4" src={image} alt={`${name} image`} loading="lazy" /> </div> <div className="flex-1 px-5 flex justify-between items-center bg-main"> <span className="block text-xl overflow-hidden text-ellipsis max-w-[80%]"> {name} </span> <Button className={cn( 'rounded-xl border border-gray-900 cursor-pointer p-2 transition hover:bg-gray active:bg-gray', )} rounded > <Link to={link}> <img src={icon} className="w-[40px] h-[40px]" alt="add icon" loading="lazy" /> </Link> </Button> </div> </div> </> ); };

**src\components\layout\Footer.tsx**

import React from 'react'; import { Link } from 'react-router-dom'; import Icon from '/icon.svg'; export const Footer = (): React.JSX.Element => { return ( <footer className="footer"> <div className="footer\_\_img"> <img className="footer\_\_img--content" src={Icon} alt="logo" /> </div> <div className="flex-grow flex gap-4 justify-between"> <div className="footer\_\_links flex flex-col md:flex-row lg:flex-row gap-2"> <Link to="/" className="link"> Главная </Link> <Link to="/register" className="link"> Регистрация </Link> <Link to="/sertificates" className="link"> Купоны и бонусы </Link> </div> <div>2025 📚</div> </div> </footer> ); };

**src\components\layout\Header.tsx**

import React, { use } from 'react'; import { Link, useLocation } from 'react-router-dom'; import Icon from '/icon.svg'; import SearchIcon from '/images/icons/search.svg'; import AccountIcon from '/images/icons/account.svg'; export const Header = (): React.JSX.Element => { const [isDropdownOpen, setIsDropdownOpen] = React.useState(false); const pathname = useLocation().pathname; return ( <header className="border-b flex justify-between"> <Link to="/" className=""> <img src={Icon} alt="Elib logo" /> </Link> <section className="flex-grow flex justify-around md:justify-around lg:justify-between items-center gap-2 px-2"> {pathname !== '/login' && pathname !== '/register' && ( <> <form onSubmit={e => e.preventDefault()} className="flex-grow flex max-w-[500px] items-center gap-2" > <input name="search" className="flex-grow w-full border rounded py-3 px-5 bg-gray " type="text" placeholder="Поиск" spellCheck /> <button type="submit" className="py-3 px-3 w-full h-full max-w-[50px] border bg-gray rounded hover:bg-gray-300" > <img src={SearchIcon} alt="search icon" /> </button> </form> <nav className="relative"> <div className="bg-gray py-3 px-3 border rounded hover:bg-gray-300 cursor-pointer" onClick={() => setIsDropdownOpen(!isDropdownOpen)} > <img src={AccountIcon} alt="user icon" /> </div> <ul className={`absolute right-0 mt-2 w-48 bg-gray border rounded shadow-lg ${ isDropdownOpen ? 'block' : 'hidden' }`} > <li> <Link to="/user" className="block px-4 py-2 hover:bg-gray-100" onClick={() => setIsDropdownOpen(false)} > user </Link> </li> <li> <Link to="/certificates" className="block px-4 py-2 hover:bg-gray-100" onClick={() => setIsDropdownOpen(false)} > Сертификаты </Link> </li> <li> <hr className="border-t" /> </li> <li> <Link to="/logout" className="block px-4 py-2 hover:bg-gray-100" onClick={() => setIsDropdownOpen(false)} > Выйти </Link> </li> </ul> </nav> </> )} {(pathname === '/login' || pathname === '/register') && ( <h1 className="text-2xl">Elib - онлайн библиотека</h1> )} </section> </header> ); };

**src\components\pages\AdminPage.tsx**

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 handleEditAndAdd = () => { setSelectedItem(undefined); console.log('deleted'); }; 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 handleEditAndAdd={handleEditAndAdd} currentList={currentList} initialValue={selectedItem} /> </section> </div> )} </main> ); }; export default AdminPage;

**src\components\pages\BookPage.tsx**

import React from 'react'; import { useSearchParams } from 'react-router-dom'; import { ReactReader } from 'react-reader'; export const BookPage = (): React.JSX.Element => { const [searchParams] = useSearchParams(); const [location, setLocation] = React.useState<string | number>(0); const bookPath = `/${searchParams.get('book')?.trim().replace('-', '')}.epub`; console.log(bookPath); return ( <main className="flex-1"> <div style={{ height: '100vh' }}> <ReactReader url={bookPath ?? ''} location={location} locationChanged={(epubcfi: string) => setLocation(epubcfi)} /> </div> </main> ); };

**src\components\pages\CertificatesPage.tsx**

import React from 'react'; import { useElibStore } from '../../store/store'; import { CertificateCard } from '../features/CertificateCard/CertificateCard'; export const CertificatesPage = (): React.JSX.Element => { const certificates = useElibStore(store => store.certificates); React.useEffect(() => { if (certificates.length === 0) { useElibStore.getState().fetchCertificates(); } }, []); return ( <main className="flex-1 p-5"> <div className="flex justify-center p-4"> <div className="flex items-center justify-between border rounded p-5"> <h2 className="text-center text-4xl">Дарите знания!</h2> <div className="overflow-hidden"> <img className="w-full" src="/images/posters/book-present.svg" alt="banner logo" /> </div> </div> </div> <div className="grid md:grid-cols-2 gap-4 p-4"> {certificates.map(certificate => ( <CertificateCard key={certificate.id} name={certificate.name} text={certificate.text} img={certificate.img} /> ))} </div> </main> ); };

**src\components\pages\LoginPage.tsx**

import React from 'react'; import { useForm } from 'react-hook-form'; import { Link } from 'react-router-dom'; import { Button } from '../ui/Button'; type FormData = { email: string; password: string; }; export const LoginPage = (): React.JSX.Element => { const { register, handleSubmit, formState: { errors }, reset, } = useForm<FormData>(); const onSubmit = (data: FormData) => { console.log(data); reset(); }; return ( <main className="flex-1 grid sm:grid-cols-1 md:grid-cols-2 gap-4 p-4 items-center justify-center"> <section className="p-4 flex justify-center"> <div className="border rounded p-4 flex flex-col gap-2 px-4 object-contain"> <h2 className="text-center text-2xl">С Возвращением</h2> <form className="flex flex-col gap-2 px-4" onSubmit={handleSubmit(data => onSubmit(data))} > <input id="email" type="email" className="border rounded auth-input" autoComplete="email" placeholder="Email" required {...register('email', { required: 'Email is required', pattern: { value: /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/, message: 'Invalid email address', }, })} /> {errors.email && ( <span className="text-red-600">{errors.email.message}</span> )} <input id="password" type="password" className="border rounded auth-input" autoComplete="password" placeholder="Пароль" required {...register('password', { required: 'password is required', })} /> {errors.password && ( <span className="text-red-600">{errors.password.message}</span> )} <Button type="submit" className="rounded-xl cursor-pointer p-2 transition active:bg-gray-700" > Войти </Button> </form> <Link to={'/register'} className="text-center text-gray-700"> <div>Впервые у нас?</div> </Link> </div> </section> <section className="hidden sm:hidden md:block"> <img src="/images/posters/book-shkaf.svg" /> </section> </main> ); };

**src\components\pages\MainPage.tsx**

import React from 'react'; import { useElibStore } from '../../store/store'; import { ProductCard } from '../features/ProductCard/ProductCard'; import type { Book } from '../../types'; export const MainPage = (): React.JSX.Element => { const books = useElibStore(store => store.books); const booksByGenre = React.useMemo(() => { console.log('Recalculating booksByGenre, books:', books); if (!books || books.length === 0) { return {}; } return books.reduce((acc: Record<string, Book[]>, book: Book) => { if (!Array.isArray(book.genres) || book.genres.length === 0) { return acc; } book.genres.forEach(genre => { if (!genre?.name) { return; } if (!acc[genre.name]) { acc[genre.name] = []; } acc[genre.name].push(book); }); return acc; }, {}); }, [books]); React.useEffect(() => { const unsub = () => { if (books.length === 0) { const addBooks = useElibStore.getState().fetchBooks; addBooks(); } }; return unsub; }, []); return ( <main className="flex-1 p-4"> {Object.entries(booksByGenre).map(([genre, books]) => ( <React.Fragment key={genre}> <div className="main-content\_\_heading-block"> <h2 className="text-3xl">{genre}</h2> <hr /> </div> <div className="main-content\_\_books"> {books.map(book => ( <ProductCard key={book.id} name={book.name} image={book.cover} link={`/books/${book.id}`} icon="/images/icons/add.svg" /> ))} </div> </React.Fragment> ))} </main> ); };

**src\components\pages\RegisterPage.tsx**

import React from 'react'; import { useForm } from 'react-hook-form'; import { Link } from 'react-router-dom'; import { Button } from '../ui/Button'; type FormData = { email: string; password: string; card: string; name: string; }; export const RegisterPage = (): React.JSX.Element => { const { register, handleSubmit, formState: { errors }, reset, } = useForm<FormData>(); const onSubmit = (data: FormData) => { console.log(data); reset(); }; return ( <main className="flex-1 grid sm:grid-cols-1 md:grid-cols-2 gap-4 p-4 items-center justify-center"> <section className="p-4 flex justify-center"> <div className="border rounded p-4 flex flex-col gap-2 px-4 object-contain"> <h2 className="text-center text-2xl">Добро пожаловать</h2> <form className="flex flex-col gap-2 px-4" onSubmit={handleSubmit(data => onSubmit(data))} > <label className="flex flex-col"> <span className="text-gray-700">Введите имя:</span> <input id="name" type="text" className="border rounded auth-input" autoComplete="name" placeholder="Имя" required {...register('name', { required: 'Name is required', })} /> {errors.name && ( <span className="text-red-600">{errors.name.message}</span> )} </label> <label className="flex flex-col"> <span className="text-gray-700">Введите email:</span> <input id="email" type="email" className="border rounded auth-input" autoComplete="email" placeholder="Email" required {...register('email', { required: 'Email is required', pattern: { value: /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/, message: 'Invalid email address', }, })} /> {errors.email && ( <span className="text-red-600">{errors.email.message}</span> )} </label> <label className="flex flex-col"> <span className="text-gray-700">Введите пароль:</span> <input id="password" type="password" className="border rounded auth-input" autoComplete="password" placeholder="Пароль" required {...register('password', { required: 'password is required', })} /> {errors.password && ( <span className="text-red-600">{errors.password.message}</span> )} </label> <label className="flex flex-col"> <span className="text-gray-700">Введите номер карты:</span> <input id="card" type="text" className="border rounded auth-input" autoComplete="card" placeholder="Номер карты" required {...register('card', { required: 'Card number is required', })} /> {errors.card && ( <span className="text-red-600">{errors.card.message}</span> )} </label> <Button type="submit" className="rounded-xl border border-gray-900 cursor-pointer p-2 transition active:bg-gray-700" > Зарегистрироваться </Button> </form> <Link to={'/login'} className="text-center text-gray-700"> <div>Войти</div> </Link> </div> </section> <section className="hidden sm:hidden md:block"> <img src="/images/posters/book-shkaf.svg" /> </section> </main> ); };

**src\components\pages\UserPage.tsx**

import React from 'react'; import { ProductCard } from '../features/ProductCard/ProductCard'; export const UserPage = (): React.JSX.Element => { return ( <main className="flex-1"> <section> <div className="main-content\_\_heading-block"> <h2> Ваши книги, <span className="text-red-500">user</span> </h2> <hr /> </div> <div className="main-content\_\_books"> <ProductCard link={'/book?book=harry-potter'} image={'/images/covers/harrypotter.jpg'} icon={'/images/icons/add.svg'} name={'Гарри Поттер и философский камень'} /> </div> </section> </main> ); };

**src\components\ui\Button.tsx**

import React, { type ComponentPropsWithoutRef } from 'react'; import { cn } from '../../lib/cn'; interface ButtonProps extends ComponentPropsWithoutRef<'button'> { children: React.ReactNode; className?: string; primary?: boolean; rounded?: boolean; onClick?: () => void; type?: 'button' | 'submit'; } export const Button = ({ children, className = '', type = 'button', primary = false, rounded = false, onClick, ...props }: Readonly<ButtonProps>): React.JSX.Element => { return ( <button className={cn( 'border border-black p-2 rounded text-black' + 'transition', { 'bg-gray hover:bg-main': primary, 'bg-main hover:bg-gray': !primary, 'w-10 h-10 flex justify-center items-center': rounded, }, className, )} {...props} onClick={onClick} type={type} > {children} </button> ); };

**src\components\ui\index.tsx**

import { svelteHoc } from '../../lib/svelteHoc'; import Button from './Button.svelte'; export const UiButton = svelteHoc(Button);

**src\index.css**

@import "tailwindcss"; @import url('https://fonts.googleapis.com/css2?family=Inter:ital,opsz,wght@0,14..32,100..900;1,14..32,100..900&display=swap'); body { min-height: 100vh; font-family: 'Inter', sans-serif; display: flex; flex-direction: column; #root { flex: 1; display: flex; flex-direction: column; } } @theme { --font-display: 'Inter', 'sans'; --color-main: #f7f7f7; --color-gray: #d9d9d9; --color-shadow: #00000025; --color-dark: #000; --color-gray-text: #707070; --rounded-main: 10px; --color-gray-footer: #8b8989; } :root { --main-color: #f7f7f7; --gray-color: #d9d9d9; --shadow-color: #00000025; --dark-color: #000; --gray-text-color: #707070; --border-radius: 10px; --gray-footer-color: #8b8989; } .link { text-decoration: none; color: #000; text-align: center; &:hover { text-decoration: underline; color: fuchsia; } &:active { text-decoration: underline; color: fuchsia; } } .max-width-500 { max-width: 500px; } .btn-primary { background-color: var(--main-color); border: 1px solid var(--dark-color); border-radius: var(--border-radius); padding: 10px 20px; max-height: fit-content; color: var(--dark-color); cursor: pointer; &:hover { background-color: var(--gray-color); border: 1px solid var(--dark-color); color: var(--dark-color); } &:active { background-color: var(--gray-color); color: var(--dark-color); } } .btn-secondary { background: var(--gray-color); border-radius: var(--border-radius); border: 1px solid var(--dark-color); cursor: pointer; width: max-content; aspect-ratio: 1; padding: 11px; font-size: medium; font-family: 'Inter', sans-serif; &:hover { background: var(--shadow-color); border: 1px solid var(--dark-color); } &:active { background: var(--shadow-color); border: 1px solid var(--dark-color); } @media screen and (width <= 420px) { font-size: small; padding: 7px; } } .form-input { background-color: var(--gray-color); border-radius: var(--border-radius); border: 1px solid var(--dark-color); padding: 10px 20px; max-height: fit-content; color: var(--dark-color); cursor: pointer; @media screen and (width <= 420px) { font-size: small; width: 3em; } } .footer { background-color: var(--main-color); border-top: 2px solid var(--dark-color); display: flex; align-items: center; gap: 30px; &\_\_links { > a { display: block; text-decoration: none; color: var(--dark-color); } } } .rounded { border-radius: var(--border-radius) !important; } .grid-rows-1 { grid-template-rows: 1fr; } .grid-cols-1 { grid-template-columns: 1fr; } .grid-md-cols-2 { @media (min-width: 768px) { grid-template-columns: repeat(2, 1fr); } } .auth-input { padding: 5px; &::placeholder { color: var(--gray-text-color); padding-left: 10px; } } .fit-content { max-width: fit-content; } .main-content { flex: 1; padding: 20px; } .main-content\_\_heading-block { margin: 20px; } .main-content\_\_books { display: grid; gap: 30px; grid-template-columns: repeat(auto-fill, minmax(250px, 1fr)); justify-items: center; align-items: center; }

**src\lib\cn.tsx**

import { clsx, type ClassValue } from 'clsx'; import { twMerge } from 'tailwind-merge'; export function cn(...inputs: ClassValue[]) { return twMerge(clsx(inputs)); }

**src\lib\config.ts**

export class Config { private \_apiUrl?: string; public loadUrl(): string { this.\_apiUrl = import.meta.env.VITE\_API\_URL; if (!this.\_apiUrl) { throw new Error('Cannot read .env file'); } return this.\_apiUrl; } get apiUrl(): string { if (!this.\_apiUrl) { throw new Error('API URL is not set. Call loadUrl() first.'); } return this.\_apiUrl; } }

**src\lib\consts.ts**

export const SERVER\_URL = 'http://localhost:3000';

**src\lib\svelteHoc.tsx**

import { useLayoutEffect, useRef } from 'react'; import { mount } from 'svelte'; export const svelteHoc = (SvelteComponent: any) => { return (props: any) => { const svelteRef = useRef<HTMLDivElement>(null); const componentRef = useRef<any>(null); const initialRenderRef = useRef(true); // We only mount once and never remount useLayoutEffect(() => { if (!svelteRef.current) return; // Clear the target element before mounting to handle StrictMode double mounting if (svelteRef.current.firstChild) { svelteRef.current.innerHTML = ''; } // Create a new props object that will be updated // Svelte 5 components will re-render when these objects are modified const mutableProps = { ...props }; // Mount the Svelte 5 component with the mutable props reference componentRef.current = mount(SvelteComponent, { target: svelteRef.current, props: mutableProps, }); // Store the mutable props for future updates componentRef.current.props = mutableProps; // Cleanup on unmount return () => { componentRef.current = null; }; }, []); // Handle prop changes without remounting useLayoutEffect(() => { // Skip the initial render since we already passed props if (initialRenderRef.current) { initialRenderRef.current = false; return; } // Skip if component isn't mounted yet if (!componentRef.current || !componentRef.current.props) return; // Update all props directly in the mutable object // This takes advantage of Svelte 5's reactivity system const mutableProps = componentRef.current.props; // First, remove any props that no longer exist Object.keys(mutableProps).forEach(key => { if (!(key in props)) { delete mutableProps[key]; } }); // Then update or add all current props Object.keys(props).forEach(key => { mutableProps[key] = props[key]; }); }, [props]); return <div ref={svelteRef} />; }; };

**src\main.tsx**

import { lazy, StrictMode } from 'react'; import { createRoot } from 'react-dom/client'; import './index.css'; import { App } from './App.tsx'; import { createBrowserRouter } from 'react-router'; import { RouterProvider } from 'react-router-dom'; import { MainPage } from './components/pages/MainPage.tsx'; import { CertificatesPage } from './components/pages/CertificatesPage.tsx'; import { RegisterPage } from './components/pages/RegisterPage.tsx'; import { LoginPage } from './components/pages/LoginPage.tsx'; import { UserPage } from './components/pages/UserPage.tsx'; import { BookPage } from './components/pages/BookPage.tsx'; const AdminPage = lazy(() => import('./components/pages/AdminPage.tsx')); const routes = createBrowserRouter([ { path: '/', element: <App />, children: [ { path: '/', element: <MainPage />, }, { path: '/certificates', element: <CertificatesPage />, }, { path: '/register', element: <RegisterPage />, }, { path: '/login', element: <LoginPage />, }, { path: '/user', element: <UserPage />, }, { path: '/book', element: <BookPage />, }, { path: '/admin', element: <AdminPage />, }, ], errorElement: ( <div> Error... <img src="https://c.tenor.com/m4Gn7YI8y4YAAAAC/tenor.gif" /> </div> ), }, ] as const); createRoot(document.getElementById('root')!).render( <StrictMode> <RouterProvider router={routes} /> </StrictMode>, );

**src\store\slices\authorsSlice.ts**

import type { StateCreator } from 'zustand'; import type { Author } from '../../types'; import { devtools } from 'zustand/middleware'; import { Store } from '../store'; import { getAuthors } from '../../api/api'; export type AuthorsSlice = { authors: Author[]; fetchAuthors: () => Promise<void>; }; export const createAuthorsSlice: StateCreator< Store, [], [['zustand/devtools', never]], AuthorsSlice > = devtools( set => ({ authors: [], fetchAuthors: async () => { const data = await getAuthors(); set({ authors: data }); }, }), { name: 'authors-slice', }, );

**src\store\slices\booksSlice.ts**

import type { StateCreator } from 'zustand'; import type { Book } from '../../types'; import { deleteBook, getBooks, postBook, updateBook as updateBookAPI, } from '../../api/api'; import { devtools } from 'zustand/middleware'; import { Store } from '../store'; export type BooksSlice = { books: Book[]; fetchBooks: () => Promise<void>; addBook: (book: FormData) => Promise<Book>; updateBook: (id: number, book: FormData) => Promise<void>; removeBook: (id: number) => Promise<void>; }; export const createBooksSlice: StateCreator< Store, [], [['zustand/devtools', never]], BooksSlice > = devtools( (set, get) => ({ books: [], fetchBooks: async () => { const data = await getBooks(); set({ books: data }); }, addBook: async (book: FormData) => { console.log('adding new book', Object.entries(book)); const data = Object.fromEntries(book) as any; const newBook: Book = { id: data.id, name: data.name, authors: JSON.parse(data.authors), genres: JSON.parse(data.genres), cover: data.poster.name, path: data.file.name, }; set(state => ({ books: [...state.books, newBook], })); // await get().fetchBooks(); // await postBook(book); return Promise.resolve(newBook); }, updateBook: async (id: number, book: FormData) => { const data = Object.fromEntries(book) as any; set(state => ({ books: state.books.map(b => { if (b.id === id) { const newBookVersion: Book = { id: id, name: data.name || b.name, authors: JSON.parse(data.authors) || b.authors, genres: JSON.parse(data.genres) || b.genres, cover: data.poster || b.cover, path: data.file || b.path, }; return newBookVersion; } return b; }), })); // await updateBookAPI(book); }, removeBook: async (id: number) => { set(state => ({ books: state.books.filter(b => b.id !== id), })); // await deleteBook(id.toFixed()); }, }), { name: 'books-slice', }, );

**src\store\slices\certificatesSlice.ts**

import type { StateCreator } from 'zustand'; import type { Certificate } from '../../types'; import { devtools } from 'zustand/middleware'; import { Store } from '../store'; import { getCertificates, postCertificate, deleteCertificate, updateCertificate, } from '../../api/api'; export type CertificatesSlice = { certificates: Certificate[]; fetchCertificates: () => Promise<void>; addCertificate: (certificate: FormData) => Promise<Certificate>; updateCertificate: (id: number, certificate: FormData) => Promise<void>; removeCertificate: (id: number) => Promise<void>; }; export const createCertificatesSlice: StateCreator< Store, [], [['zustand/devtools', never]], CertificatesSlice > = devtools( set => ({ certificates: [], fetchCertificates: async () => { const data = await getCertificates(); set({ certificates: data }); }, addCertificate: async (certificate: FormData) => { const newCertificate = await postCertificate(certificate); set(state => ({ certificates: [...state.certificates, newCertificate], })); return newCertificate; }, updateCertificate: async (id: number, certificate: FormData) => { const data = await updateCertificate(certificate); set(state => ({ certificates: state.certificates.map(c => (c.id === id ? data : c)), })); }, removeCertificate: async (id: number) => { set(state => ({ certificates: state.certificates.filter(c => c.id !== id), })); await deleteCertificate(id.toFixed()); }, }), { name: 'certificates-slice', }, );

**src\store\slices\genresSlice.ts**

import type { StateCreator } from 'zustand'; import type { Genre } from '../../types'; import { devtools } from 'zustand/middleware'; import { Store } from '../store'; import { getGenres } from '../../api/api'; export type GenresSlice = { genres: Genre[]; fetchGenres: () => Promise<void>; isGenresLoaded: boolean; }; export const createGenresSlice: StateCreator< Store, [], [['zustand/devtools', never]], GenresSlice > = devtools( set => ({ isGenresLoaded: false, genres: [], fetchGenres: async () => { set({ isGenresLoaded: false }); const data = await getGenres(); set({ genres: data, isGenresLoaded: true }); }, }), { name: 'genres-slice', }, );

**src\store\slices\sharedSlice.ts**

import type { StateCreator } from 'zustand'; import { Store } from '../store'; export type SharedSlice = { getBoth: () => Promise<void>; }; export const createSharedSlice: StateCreator<Store, [], [], SharedSlice> = ( \_, get, ) => ({ getBoth: async () => { const store = get(); await store.fetchAuthors(); await store.fetchGenres(); await store.fetchBooks(); await store.fetchCertificates(); }, });

**src\store\store.ts**

import { create } from 'zustand'; import { subscribeWithSelector } from 'zustand/middleware'; // Импортируем типы слайсов import type { AuthorsSlice } from './slices/authorsSlice'; import type { GenresSlice } from './slices/genresSlice'; import type { BooksSlice } from './slices/booksSlice'; import type { CertificatesSlice } from './slices/certificatesSlice'; import type { SharedSlice } from './slices/sharedSlice'; // Определяем тип Store как комбинацию всех слайсов export type Store = AuthorsSlice & GenresSlice & BooksSlice & CertificatesSlice & SharedSlice; // Импортируем функции создания слайсов import { createAuthorsSlice } from './slices/authorsSlice'; import { createGenresSlice } from './slices/genresSlice'; import { createBooksSlice } from './slices/booksSlice'; import { createCertificatesSlice } from './slices/certificatesSlice'; import { createSharedSlice } from './slices/sharedSlice'; export const useElibStore = create<Store>()( subscribeWithSelector((...args) => ({ ...createAuthorsSlice(...args), ...createGenresSlice(...args), ...createBooksSlice(...args), ...createCertificatesSlice(...args), ...createSharedSlice(...args), })), );

**src\types.d.ts**

export interface Book { id: number; name: string; authors: Author[]; genres: Genre[]; cover: string; path: string; } export interface Certificate { id: number; name: string; text: string; img: string; } export interface Genre { id: number; name: string; } export interface Author { id: number; name: string; } export interface BookFormData { id: number; name: string; authors: Author[]; selectedGenres: Genre[]; cover: string; path: string; } export type List = 'books' | 'certificates';

**src\vite-env.d.ts**

/// <reference types="vite/client" />

**tsconfig.app.json**

{ "compilerOptions": { "tsBuildInfoFile": "./node\_modules/.tmp/tsconfig.app.tsbuildinfo", "target": "ES2020", "useDefineForClassFields": true, "lib": ["ES2020", "DOM", "DOM.Iterable"], "module": "ESNext", "skipLibCheck": true, /\* Bundler mode \*/ "moduleResolution": "bundler", "allowImportingTsExtensions": true, "isolatedModules": true, "moduleDetection": "force", "noEmit": true, "jsx": "react-jsx", /\* Linting \*/ "strict": true, "noUnusedLocals": true, "noUnusedParameters": true, "noFallthroughCasesInSwitch": true, "noUncheckedSideEffectImports": true }, "include": ["src"] }

**tsconfig.json**

{ "files": [], "references": [ { "path": "./tsconfig.app.json" }, { "path": "./tsconfig.node.json" } ] }

**tsconfig.node.json**

{ "compilerOptions": { "tsBuildInfoFile": "./node\_modules/.tmp/tsconfig.node.tsbuildinfo", "target": "ES2022", "lib": ["ES2023"], "module": "ESNext", "skipLibCheck": true, /\* Bundler mode \*/ "moduleResolution": "bundler", "allowImportingTsExtensions": true, "isolatedModules": true, "moduleDetection": "force", "noEmit": true, /\* Linting \*/ "strict": true, "noUnusedLocals": true, "noUnusedParameters": true, "noFallthroughCasesInSwitch": true, "noUncheckedSideEffectImports": true }, "include": ["vite.config.ts"] }

**vite.config.ts**

import { defineConfig } from 'vite'; import react from '@vitejs/plugin-react'; import tailwindcss from '@tailwindcss/vite'; import { svelte } from '@sveltejs/vite-plugin-svelte'; // https://vite.dev/config/ export default defineConfig({ plugins: [ react(), tailwindcss(), svelte({ compilerOptions: { runes: true, }, }), ], });