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 № 4**

AJAX, REST API, Компоненты на JS

Выполнил:

студент гр. ПИбд-23

Пазушкин И.П.

Проверил:

преподаватель

Истюков Т.А.

Ульяновск,   
2025 г.

**Задание лабораторной работы**

1. В рамках данной лабораторной работы необходимо разработать компонент на чистом JS (pure JS) для отображения, редактирования и удаления сущностей некоторой предметной области (см. ЛР 3). Компонент должен состоять из трех модулей: контроллер, модель, представление. При разработке компонента необходимо придерживаться архитектуры MVC.

2. Представление компонента может быть сформировано на основе шаблона (HTML Template) или сгенерировано с помощью методов JS (createElement и т. д.).

3. Shadow DOM можно использовать на свое усмотрение.

4. Для работы компонента необходимо реализовать прототип серверной части приложения с REST API на основе json-server. Модель данных БД json-server должна содержать не менее трех сущностей: одна основная, две вспомогательные. Между основной и вспомогательными сущностями должна существовать связь «один-ко-многим» (на уровне id). Для работы с json-server следует внести изменения в файл package.json.

5. Взаимодействие между клиентом и сервером необходимо реализовать с помощью асинхронных функций (async/await) и методов Fetch API.

6. При разработке ЛР не следует изменять макет, полученный ранее в ЛР 3. Следует дополнить ранее созданный макет дополнительными кнопками для вызова функций создания, редактирования и удаления сущностей. Для создания и редактирования сущностей необходимо использовать форму, которая располагается на отдельной странице или в модальном окне. Располагать форму на странице со списком сущностей нельзя.

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

**Порядок выполнения работы**

Мной был изучен пример преподавателя: я выполнял работу, используя и изменяя под себя его архитектуру MVC, эмуляцию REST API и общую логику работы с сервером. Так у меня получилась следующая структура:  
**1. Архитектура MVC**

Модель (Model):

* Реализована в файлах model.ts для карточки и формы (AdminForm).
* Отвечает за работу с данными: получение, обновление, удаление, а также за взаимодействие с сервером через REST API (используя fetch).

Представление (View):

* Реализована в файлах view.ts для AdminView.
* Формирует интерфейс динамически с помощью createElement (для карточек) и на основе шаблона HTML (для формы).

Контроллер (Controller):

* Связывает модель и представление, обрабатывает пользовательские действия (клики, отправку формы).
* Реализован в файлах controller.ts для карточки и формы (AdminForm).

**2. Работа с сервером**

Использован express для REST API. Модель данных включает четыре сущности:

* Основная: книги.
* Вспомогательные: жанры и авторы.
* Дополнительная: подарочные сертификаты.

Взаимодействие с сервером реализовано через асинхронные функции и Fetch API (файл client.ts).

**3. Функциональность**

Каталог (admin.html):

* Отображение карточек книг с названиями.
* Возможности для карточек: добавление, редактирование и удаление.
* Кнопка для перехода к созданию новой записи.

Форма (admin.html):

* Создание и редактирование записей с валидацией полей (цена не может быть отрицательной, проверка изображения на формат).
* Динамическая подгрузка списков групп и поставщиков.
* Отображение успешного сохранения через alert-уведомления.

**Скриншоты**

*Страница admin.html:*

*Добавление:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Изменение:*

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

Контент, сгенерированный ИИ, может содержать ошибки.*

*Удаление:*

*Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Контент, сгенерированный ИИ, может содержать ошибки.*

**Исходный код**

**certificates.json**

[ { "id": 1, "name": "купон1", "text": "текст", "img": "url" } ]

**eslint.config.js**

import perfectionistPlugin from 'eslint-plugin-perfectionist'; import stylisticJsPlugin from '@stylistic/eslint-plugin-js'; import tseslint from '@typescript-eslint/eslint-plugin'; import prettierPlugin from 'eslint-plugin-prettier'; import tsParser from '@typescript-eslint/parser'; import jsPlugin from '@eslint/js'; import globals from 'globals'; export default [ jsPlugin.configs.recommended, perfectionistPlugin.configs['recommended-line-length'], { rules: { 'stylistic/padding-line-between-statements': [ 'error', { blankLine: 'always', prev: 'import', next: '\*' }, { blankLine: 'any', prev: 'import', next: 'import' }, { blankLine: 'always', next: 'return', prev: '\*' }, { blankLine: 'always', prev: 'block-like', next: '\*' }, { blankLine: 'always', next: 'block-like', prev: '\*' }, { blankLine: 'always', prev: 'export', next: '\*' }, { blankLine: 'always', next: 'export', prev: '\*' }, { prev: 'singleline-const', blankLine: 'always', next: '\*' }, { prev: 'singleline-let', blankLine: 'always', next: '\*' }, { next: 'singleline-const', blankLine: 'always', prev: '\*' }, { next: 'singleline-let', blankLine: 'always', prev: '\*' }, { prev: 'singleline-let', next: 'singleline-let', blankLine: 'any' }, { prev: 'singleline-const', next: 'singleline-const', blankLine: 'any', }, ], 'prettier/prettier': [ 'error', { jsxSingleQuote: false, endOfLine: 'auto', singleQuote: true, }, ], 'no-promise-executor-return': 'error', 'array-callback-return': 'error', 'no-unused-expressions': 'error', 'prefer-arrow-callback': 'error', 'prefer-destructuring': 'error', 'consistent-return': 'error', 'arrow-body-style': 'error', 'object-shorthand': 'error', 'no-return-assign': 'error', 'no-await-in-loop': 'error', 'no-throw-literal': 'error', 'no-extend-native': 'error', 'no-return-await': 'error', 'prefer-template': 'error', 'no-else-return': 'error', 'accessor-pairs': 'error', 'no-lone-blocks': 'error', 'require-await': 'error', 'prefer-const': 'error', 'dot-notation': 'error', 'no-multi-str': 'error', 'camelcase': 'error', 'no-proto': 'error', 'curly': 'error', }, languageOptions: { globals: { ...globals.browser, ...globals.es2025, ...globals.jest, }, }, plugins: { stylistic: stylisticJsPlugin, prettier: prettierPlugin, }, files: ['\*\*/\*.{js,ts}'], }, { rules: { ...tseslint.configs.recommended.rules, '@typescript-eslint/explicit-function-return-type': 'off', '@typescript-eslint/no-explicit-any': 'warn', '@typescript-eslint/no-unused-vars': 'warn', }, languageOptions: { parserOptions: { project: './tsconfig.json', }, parser: tsParser, }, plugins: { '@typescript-eslint': tseslint, }, files: ['\*\*/\*.ts'], }, { ignores: ['node\_modules', 'vite.config.js', 'dist'], }, ];

**package.json**

{ "name": "elib", "version": "0.0.0", "devDependencies": { "@eslint/js": "^9.20.0", "@stylistic/eslint-plugin-js": "^3.1.0", "@types/bootstrap": "^5.2.10", "@types/cors": "^2.8.17", "@types/express": "^5.0.0", "@types/express-fileupload": "^1.5.1", "@types/node": "^22.13.10", "@typescript-eslint/eslint-plugin": "^8.26.0", "@typescript-eslint/parser": "^8.26.0", "eslint-plugin-perfectionist": "^4.8.0", "eslint-plugin-prettier": "^5.2.3", "globals": "^15.14.0", "vite": "^6.1.0" }, "private": true, "scripts": { "dev": "vite", "build": "vite build", "preview": "vite preview", "lint": "eslint . --ext .js,.ts", "lint:fix": "eslint . --ext .js,.ts --fix", "format": "prettier --write .", "fix": "bun lint:fix && bun format", "server": "bun run server/index.ts" }, "type": "module", "dependencies": { "@popperjs/core": "^2.11.8", "@types/multer": "^1.4.12", "bootstrap": "^5.3.3", "bootstrap-icons": "^1.11.3", "cors": "^2.8.5", "epubjs": "^0.3.93", "express": "^4.21.2", "express-fileupload": "^1.5.1", "json-server": "^1.0.0-beta.3", "multer": "^1.4.5-lts.2", "sass": "^1.85.1", "typescript": "^5.8.2" } }

**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, '..', 'src', 'public', 'books.json', ); export const CERTIFICATES\_PATH = path.join( \_\_dirname, '..', 'src', 'public', 'certificates.json', ); export const GENRES\_PATH = path.join( \_\_dirname, '..', 'src', 'public', 'genres.json', ); export const AUTHORS\_PATH = path.join( \_\_dirname, '..', 'src', 'public', 'authors.json', ); export interface Book { authors: Author[]; genres: Genre[]; cover: string; name: string; path: string; id: number; } export interface Certificate { name: string; text: string; img: string; id: number; } export interface Author { name: string; id: number; } export interface Genre { name: string; id: number; }

**server\index.ts**

import express, { RequestHandler, Response, Request } from 'express'; import fileUpload from 'express-fileupload'; import multer from 'multer'; import cors from 'cors'; import path from 'path'; import { CERTIFICATES\_PATH, AUTHORS\_PATH, Certificate, GENRES\_PATH, BOOKS\_PATH, Author, Genre, Book, } from './constants'; import { updateItemJson, updateJson, deleteItem, checkItem, readJson, } from './IO'; const app = express(); app.use(cors()); app.use(express.json()); app.use(fileUpload()); app.use('/public', express.static(path.join(\_\_dirname, '..', 'src', 'public'))); // Books app.get('/books', async (req, res) => { try { const books = await readJson(BOOKS\_PATH); const authors = await readJson(AUTHORS\_PATH); const genres = await readJson(GENRES\_PATH); const booksWithFullData = books.map((book: any) => ({ ...book, authors: book.authors .map((authorId: number) => authors.find((a: Author) => a.id === authorId), ) .filter(Boolean), genres: book.genres .map((genreId: number) => 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' }); } const authorIds = req.body.authorIds ? typeof req.body.authorIds === 'string' ? req.body.authorIds.split(',').map(Number) : Array.isArray(req.body.authorIds) ? req.body.authorIds.map(Number) : [] : []; const genreIds = req.body.genreIds ? typeof req.body.genreIds === 'string' ? req.body.genreIds.split(',').map(Number) : Array.isArray(req.body.genreIds) ? req.body.genreIds.map(Number) : [] : []; const authors = await readJson(AUTHORS\_PATH); const genres = await readJson(GENRES\_PATH); const bookData: Book = { authors: authors.filter((a: Author) => authorIds.includes(a.id)), genres: genres.filter((g: Genre) => genreIds.includes(g.id)), id: +(Math.random() \* 1000).toFixed(0), cover: `/images/covers/${poster.name}`, path: `/books/${file.name}`, name: req.body.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, '..', 'src', 'public', file.name)); await poster.mv( path.join( \_\_dirname, '..', 'src', '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; const authors = req.body.authorIds ? req.body.authorIds.split(',').map(Number) : existingBook.authors; const genres = req.body.genreIds ? req.body.genreIds.split(',').map(Number) : existingBook.genres; const updatedBook: Book = { cover: coverFile ? `/images/covers/${coverFile.name}` : existingBook.cover, name: req.body.name || existingBook.name, path: existingBook.path, id: bookId, authors, genres, }; if (coverFile) { await coverFile.mv( path.join( \_\_dirname, '..', 'src', 'public', 'images', 'covers', coverFile.name, ), ); } const result = await updateItemJson(updatedBook, BOOKS\_PATH); res.json(result); } catch (error) { console.error('Error updating book:', error); res.status(500).json({ error: 'Failed to update book' }); } }; app.put('/books/:id', putBookHandler); app.delete('/books/:id', async (req, res) => { try { await deleteItem(req.params.id, BOOKS\_PATH); res.status(200).json({ message: 'Книга удалена' }); } catch (e) { res.status(500).json({ message: 'Ошибка при удалении книги' }); } }); // Certificates app.get('/certificates', async (req, res) => { try { const certificates = await readJson(CERTIFICATES\_PATH); res.json(certificates); } catch (e) { res.status(500).json({ message: 'Ошибка при получении сертификатов' }); } }); app.post('/certificates', async (req, res) => { if (!req.files || !req.files.img) { return res.status(400).json({ message: 'Изображение не загружено' }); } const img = req.files.img as fileUpload.UploadedFile; if (!img.name.endsWith('.jpg') && !img.name.endsWith('.png')) { return res .status(400) .json({ message: 'Загрузите изображение в формате JPG или PNG' }); } const certData: Certificate = { img: `/images/posters/${img.name}`, name: req.body.name, text: req.body.text, id: 0, // Will be set by updateJson }; 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, '..', 'src', '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({ text: certificate.text, img: certificate.img, id: certificate.id, }); } 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, img: img ? `/images/posters/${img.name}` : existingCert.img, name: req.body.name || existingCert.name, text: req.body.text || existingCert.text, id: certId, }; if (img) { await img.mv( path.join( \_\_dirname, '..', 'src', '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(3000, () => { console.log(`Server is running on port 3000`); });

**server\IO.ts**

import fs from 'fs/promises'; import type { Certificate, Author, Genre, Book } from './constants'; export async function updateItemJson( item: Certificate | Book, path: string, ): Promise<Certificate | Book> { 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: Certificate | Book) => 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: Certificate | Book, path: string, ): Promise<Certificate | Book> { 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: Certificate | Book) => 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 deleteItem(id: string, path: string): Promise<void> { try { const data = await readJson(path); const filteredData = data.filter( (item: Certificate | Book) => 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; } } export async function readJson( path: string, ): Promise<(Certificate | Author | Genre | Book)[]> { 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: Certificate | Book, path: string, ): Promise<boolean> { const items = await readJson(path); return items.some((i: Certificate | Book) => i.name === item.name); } function getEntityType(item: Certificate | Book): string { if ('authors' in item && 'genres' in item) { return 'Book'; } if ('img' in item && 'text' in item) { return 'Certificate'; } return 'Unknown'; }

**src\admin.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <title>Админ панель</title> <link rel="icon" href="/icon.svg" /> <script src="../scripts/index.ts" type="module" defer></script> <script src="../scripts/book/bookService.ts" type="module" defer></script> <script src="../scripts/book/bookService.ts" type="module" defer></script> <script src="../scripts/admin.ts" type="module" defer></script> <link rel="stylesheet" href="../styles/style.scss" /> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li><a class="dropdown-item" href="/user">user</a></li> <li> <a class="dropdown-item" href="/sertificates">Сертификаты</a> </li> <li><hr class="dropdown-divider" /></li> <li> <a class="dropdown-item" href="/logout">Выйти</a> </li> </ul> </nav> </section> </header> <main class="main d-flex w-100 flex-column align-items-center my-xl-auto" id="root" style="min-height: calc(100vh - 150px)" > <admin-element class="w-100 h-100 flex-grow-1 d-flex align-items-center justify-content-center" ></admin-element> </main> <footer class="footer"> <div class="footer\_\_img"> <img class="footer\_\_img--content" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="footer\_\_links d-flex flex-column flex-md-row flex-lg-row gap-2" > <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**src\book.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <link rel="icon" type="image/svg" href="/icon.svg" /> <title>Книга</title> <link href="./styles/style.scss" rel="stylesheet" /> <script type="module" src="./scripts/book/book.ts" defer></script> </head> <body> <main class="book-reader d-flex align-items-center position-relative vh-100" > <button id="prev" type="button" class="btn btn-primary position-absolute start-0 top-50 translate-middle-y d-flex align-items-center justify-content-center p-3 z-3" > <img src="/images/icons/arrow-back.svg" alt="Назад" /> </button> <div id="area" class="w-100 vh-100"></div> <button id="next" type="button" class="btn btn-primary position-absolute end-0 top-50 translate-middle-y d-flex align-items-center justify-content-center p-3 z-3" > <img src="/images/icons/arrow-forward.svg" alt="Вперед" /> </button> </main> </body> </html>

**src\index.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <link rel="icon" type="image/svg" href="/icon.svg" /> <title>Elib - онлайн библиотека</title> <script src="../scripts/index.ts" type="module"></script> <script src="../scripts/book/bookFetcher.ts" type="module"></script> <link rel="stylesheet" href="../styles/style.scss" /> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li><a class="dropdown-item" href="/user">user</a></li> <li> <a class="dropdown-item" href="/sertificates">Сертификаты</a> </li> <li><hr class="dropdown-divider" /></li> <li> <a class="dropdown-item" href="/logout">Выйти</a> </li> </ul> </nav> </section> </header> <main class="main-content"></main> <footer class="footer"> <div class="footer\_\_img"> <img class="footer\_\_img--content" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="footer\_\_links d-flex flex-column flex-md-row flex-lg-row gap-2" > <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**src\login.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <link rel="icon" type="image/svg" href="/icon.svg" /> <title>Вход</title> <link rel="stylesheet" href="../styles/style.scss" /> <script src="../scripts/index.ts" type="module"></script> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li> <a class="dropdown-item" href="/register">Зарегистрироваться</a> </li> </ul> </nav> </section> </header> <main class="d-grid grid-cols-1 grid-md-cols-2 align-items-center justify-content-center gap-4" > <section class="p-4 d-flex justify-content-center"> <div class="border rounded p-4 d-flex flex-column gap-2 px-4 fit-content" > <h1 class="text-center">С возвращением!</h1> <form class="d-flex flex-column gap-2 px-4"> <!-- <label class="text-center" for="email">введите email</label> --> <input id="email" name="email" type="email" class="border rounded auth-input" autocomplete="email" placeholder="Email" required /> <!-- <label class="text-center" for="password">придумайте пароль</label> --> <input id="password" name="password" type="password" autocomplete="new-password" class="border rounded auth-input" placeholder="Пароль" required /> <div class="d-flex justify-content-center"> <button type="submit" class="btn-primary">Войти</button> </div> </form> <a class="link" href="./register">зарегистрироваться</a> </div> </section> <section class="d-none d-md-block"> <img src="/images/posters/book-shkaf.svg" /> </section> </main> <footer class="footer"> <div class="footer\_\_img"> <img class="footer\_\_img--content" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="footer\_\_links d-flex flex-column flex-md-row flex-lg-row gap-2" > <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**src\public\authors.json**

[ { "id": 0, "name": "Артур Конан Дойл" }, { "id": 1, "name": "Джоан Роулинг" } ]

**src\public\books copy.json**

[ { "id": 0, "title": "Шерлок Холмс", "author": "Артур Конан Дойл", "path": "sherlockholmes.epub", "genre": "Детектив", "cover": "/images/covers/sherlockholmes.jpg" }, { "id": 1, "title": "Гарри Поттер и философский камень", "author": "Джоан Роулинг", "path": "harrypotter.epub", "genre": "Фэнтези", "cover": "/images/covers/harrypotter.jpg" } ]

**src\public\books.json**

[ { "id": 1, "name": "Шерлок Холмс1", "authors": [ 0 ], "genres": [ 2 ], "cover": "/images/covers/sherlockholmes.jpg", "path": "sherlockholmes.epub" }, { "id": 2, "name": "Гарри Поттер и философский камень", "authors": [ 1 ], "genres": [ 1 ], "cover": "/images/covers/harrypotter.jpg", "path": "harrypotter.epub" }, { "id": 3, "name": "Назад в будущее", "authors": [ 1 ], "genres": [], "cover": "/images/covers/scorpion-fly.png", "path": "/books/Sherlock.epub" }, { "id": 735, "name": "купон2", "authors": [ 0 ], "genres": [ 1 ], "cover": "/images/covers/scorpion-stay.png", "path": "/books/Harry.epub" } ]

**src\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" }, { "id": 2, "name": "Назад в будущее", "text": "текст", "img": "/images/posters/scorpion-fly.png" } ]

**src\public\genres.json**

[ { "id": 1, "name": "Детектив" }, { "id": 2, "name": "Фэнтези" } ]

**src\public\users.json**

[ { "certificates": [], "password": "123", "email": "email@email.com", "books": [], "name": "ilya23", "card": "0000 0000 0000 0000", "id": 1 } ]

**src\register.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <link rel="icon" type="image/svg" href="/icon.svg" /> <title>Регистрация</title> <link rel="stylesheet" href="../styles/style.scss" /> <script src="../scripts/index.ts" type="module"></script> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li> <a class="dropdown-item" href="/login">Войти</a> </li> </ul> </nav> </section> </header> <main class="d-grid grid-cols-1 grid-md-cols-2 align-items-center justify-content-center gap-4" > <section class="d-flex justify-content-center p-4"> <div class="border rounded p-4 d-flex flex-column gap-2 px-4 justify-content-center" > <h1 class="text-center">Добро пожаловать!</h1> <form class="d-flex flex-column gap-2 px-4"> <label class="text-center" for="email">введите email</label> <input id="email" name="email" type="email" class="border rounded auth-input" autocomplete="email" placeholder="Email" required /> <label class="text-center" for="password">придумайте пароль</label> <input id="password" name="password" type="password" autocomplete="new-password" class="border rounded auth-input" placeholder="Пароль" required /> <label class="text-center" for="cc-number" >и отдайте все ваши деньги</label > <input id="cc-number" type="number" autocomplete="cc-number" class="border rounded auth-input" placeholder="xxxx xxxx xxxx xxxx" /> <div class="d-flex justify-content-center"> <button type="submit" class="btn-primary"> Зарегистрироваться </button> </div> </form> <a class="link" href="./login">войти</a> </div> </section> <section class="d-none d-md-block"> <img src="/images/posters/book-shkaf.svg" /> </section> </main> <footer class="footer"> <div class="footer\_\_img"> <img class="footer\_\_img--content" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="footer\_\_links d-flex flex-column flex-md-row flex-lg-row gap-2" > <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**src\scripts\admin.ts**

import { AdminController } from './components/admin/controller'; customElements.define('admin-element', AdminController);

**src\scripts\api\api.ts**

import { getAllItems, createItem, updateItem, deleteItem } from './client'; import { 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\scripts\api\client.ts**

import { SERVER\_URL } from './constants'; import { Book, Certificate } from '../../types'; 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 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.debug(path, json); return json; } catch (error) { if (error instanceof SyntaxError) { throw new Error(`There was a SyntaxError', { cause: ${error} }`); } else { throw new Error(`There was an error', { cause: ${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\scripts\api\constants.ts**

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

**src\scripts\book\book.ts**

import { BookReader } from './bookReader'; let reader: BookReader | null = null; document.addEventListener('DOMContentLoaded', async () => { console.log('[DOM] Content loaded'); try { const params = new URLSearchParams(window.location.search); const bookPath = params.get('book'); console.log('[DOM] Book path from URL:', bookPath); if (!bookPath) { throw new Error('Add ?book= parameter to URL'); } reader = new BookReader(bookPath.replace('-', '').concat('.epub')); console.log('[DOM] Reader instance created'); await reader.init(); console.log('[DOM] Reader initialization completed'); } catch (error) { console.error('[DOM Error]', error); } }); const [next, prev] = document.querySelectorAll('button'); next?.addEventListener('click', async () => { console.log('[UI] Prev button clicked'); await reader?.prevPage(); }); prev?.addEventListener('click', async () => { console.log('[UI] Next button clicked'); await reader?.nextPage(); });

**src\scripts\book\bookFetcher.ts**

import type { Book } from '../../types'; import { getGenres, getBooks } from '../api/api'; document.addEventListener('DOMContentLoaded', async () => { const books = await getBooks(); const genres = await getGenres(); console.log(books, Object.keys(books).length); if (!books || Object.keys(books).length === 0) { console.warn('Книги не найдены'); return; } const container = document.querySelector('.main-content'); const booksByGenre: Record<string, HTMLElement[]> = {}; console.log(container); books.forEach((book: Book) => { const card = document.createElement('product-card'); card.setAttribute('name', book.name); card.setAttribute('link', '/'); card.setAttribute('img', book.cover); book.genres.forEach(genre => { const genreObj = genres.find(g => g.id === genre.id); if (!genreObj) { return; } if (!booksByGenre[genreObj.name]) { booksByGenre[genreObj.name] = [card]; } else { booksByGenre[genreObj.name].push(card); } }); }); console.log(booksByGenre); Object.entries(booksByGenre).forEach(([genre, books]) => { const genreContainer = document.createElement('section'); genreContainer.innerHTML = ` <div class="main-content\_\_heading-block"> <h2>${genre}</h2> <hr /> </div> <div class="main-content\_\_books"> ${books.map(book => book.outerHTML).join('')} </div> `; console.log(genre, books); container?.appendChild(genreContainer); }); });

**src\scripts\book\bookReader.ts**

import ePub from 'epubjs'; export class BookReader { private rendition: ePub.Rendition | null; private isReady: boolean; private book: ePub.Book; constructor(bookPath: string) { console.log('[Constructor] Initializing with book path:', bookPath); this.book = ePub(bookPath); this.rendition = null; this.isReady = false; this.book.on('error', (error: unknown) => { console.error('[Book Error]', error); }); } async init() { try { console.log('[Init] Starting initialization...'); await this.book.ready; console.log('[Init] Book metadata:', (this.book as any).metadata); console.log('[Init] Creating rendition...'); console.log(document.querySelector('#area')); this.rendition = this.book.renderTo('area', { allowScriptedContent: true, height: '100%', spread: 'none', width: '100%', }); console.log('[Init] Displaying content...'); await this.rendition.display(); this.isReady = true; console.log('[Init] Rendition ready:', this.rendition); console.log('[Init] Current location:', this.rendition.currentLocation()); } catch (error) { console.error('[Init Error]', error); throw error; } } async prevPage() { console.log('[Navigation] Attempting previous page...'); if (!this.isReady) { console.warn('[Navigation] Reader not ready!'); return; } try { await this.rendition?.prev(); console.log( '[Navigation] New location:', this.rendition?.currentLocation(), ); } catch (error) { console.error('[Navigation Error]', error); } } async nextPage() { console.log('[Navigation] Attempting next page...'); if (!this.isReady) { console.warn('[Navigation] Reader not ready!'); return; } try { await this.rendition?.next(); console.log( '[Navigation] New location:', this.rendition?.currentLocation(), ); } catch (error) { console.error('[Navigation Error]', error); } } }

**src\scripts\book\bookService.ts**

const bookForm = document.getElementById('bookForm') as HTMLFormElement | null; const handleAddBook = async (event: SubmitEvent) => { event.preventDefault(); if (!bookForm) { console.error('Book form not found'); return; } const formData = new FormData(bookForm); const file = formData.get('file') as File; const poster = formData.get('poster') as File; console.log(Object.fromEntries(formData)); if (!file.name.endsWith('.epub')) { alert('Пожалуйста, загрузите файл в формате EPUB'); return; } if (!(poster.name.endsWith('.jpg') || poster.name.endsWith('.png'))) { alert('Пожалуйста, загрузите обложку в формате JPG или PNG'); return; } const response = await fetch('http://localhost:3000/add-book', { method: 'POST', body: formData, }); if (!response.ok) { console.warn('Ошибка при загрузке книги', await response.text()); return; } console.log('Книга успешно загружена', await response.json()); alert('Книга успешно загружена'); }; bookForm?.addEventListener('submit', handleAddBook);

**src\scripts\certificates\certificatesFetcher.ts**

import { getCertificates } from '../api/api'; document.addEventListener('DOMContentLoaded', async () => { const certificates = await getCertificates(); console.log(certificates, Object.keys(certificates).length); if (!certificates || Object.keys(certificates).length === 0) { console.warn('Книги не найдены'); return; } const container = document.querySelector('#certificates-list'); Object.entries(certificates).forEach(([, data]) => { const certificatesContainer = document.createElement('section'); certificatesContainer.innerHTML = ` <section class="col d-flex justify-content-center border rounded p-4"> <div class="d-flex justify-content-center"> <img class="w-100" src="${data.img}" alt="banner logo" /> </div> <div class="d-flex flex-column gap-2 align-items-center justify-content-center" > <h3 class="text-center">${data.name}</h3> <span class="text-center" >${data.text}</span > <button type="button" class="btn-primary">Купить</button> </div> </section> `; container?.appendChild(certificatesContainer); }); });

**src\scripts\components\admin\controller.ts**

import { AdminModel } from './model'; import { AdminView } from './view'; export class AdminController extends HTMLElement { private model: AdminModel; private view: AdminView; constructor() { super(); const toggleCategoryCallback = this.toggleCategory.bind(this); const deleteCallback = this.deleteItem.bind(this); const selectCallback = this.selectItem.bind(this); const submitCallback = this.submitForm.bind(this); this.model = new AdminModel(); this.view = new AdminView( this, this.model, toggleCategoryCallback, deleteCallback, selectCallback, submitCallback, ); } async submitForm(data: FormData, isUpdate: boolean) { const category = this.model.currentCategory; this.model.addItemOptimistic(data, category); this.view.update(); try { if (isUpdate) { await this.model.updateItem(data, category); } else { await this.model.addItem(data, category); } await this.getAllItems(); } catch (err) { console.error('Ошибка операции:', err); this.model.rollbackOptimistic(category); this.view.update(); } } async deleteItem(id: number) { const category = this.model.currentCategory; this.model.deleteItemOptimistic(id, category); this.view.update(); try { await this.model.deleteItem(id, category); await this.getAllItems(); } catch (err) { console.error('Ошибка удаления:', err); this.model.rollbackOptimistic(category); this.view.update(); } } toggleCategory(category: string) { this.model.currentCategory = category as 'certificates' | 'books'; this.view.update(); } selectItem(id: number) { this.model.selectedId = id; this.view.update(); } async getAllItems() { await this.model.fetchAll(); this.view.update(); } connectedCallback() { this.view.render(); this.getAllItems(); } }

**src\scripts\components\admin\list.ts**

import { Certificate, Book } from '../../../types'; export class ItemsList extends HTMLElement { public set items(value: (Certificate | Book)[]) { this.\_listItems = value; this.update(); } public get items(): (Certificate | Book)[] { return this.\_listItems; } public get activeItem(): number { return this.\_active; } private \_listItems: (Certificate | Book)[] = []; private \_active: number = -1; constructor() { super(); } render(): string { if (!this.\_listItems || this.\_listItems.length === 0) { return ` <style> .list-container { max-height: 400px; overflow-y: auto; } </style> <div class="list-container"> <div class="list-group" id="list-tab"> <p>Список пуст</p> </div> </div> `; } const alphabetSorted = this.\_listItems.sort((a, b) => a.name.localeCompare(b.name, 'ru', { sensitivity: 'base', caseFirst: 'upper', }), ); console.log('sorted', alphabetSorted, this.\_listItems); return ` <style> .list-container { max-height: 400px; overflow-y: auto; } .list-group-item { display: flex; justify-content: space-between; align-items: center; padding: 10px; border: 1px solid #ddd; margin-bottom: 5px; cursor: pointer; } .list-group-item.active { background-color: #d9d9d9; color: #000; border-color: #d9d9d9; } .delete-btn { cursor: pointer; color: red; font-weight: bold; } </style> <div class="list-container"> <div class="list-group" id="list-tab"> ${alphabetSorted .map( (elem, index) => ` <div class="list-group-item list-group-item-action ${index === this.\_active ? 'active' : ''}" id="list-${index}-list" data-id="${elem.id}"> <span>${elem.name}</span> <span class="delete-btn" data-id="${elem.id}">×</span> </div> `, ) .join('')} </div> </div>`; } addEventListeners(): void { const listItems = this.shadowRoot!.querySelectorAll('.list-group-item'); listItems.forEach((item, index) => { item.addEventListener('click', e => { if ((e.target as HTMLElement).classList.contains('delete-btn')) { return; } listItems.forEach(elem => elem.classList.remove('active')); item.classList.add('active'); this.\_active = index; this.dispatchEvent( new CustomEvent('active-changed', { detail: { active: this.\_active }, composed: true, bubbles: true, }), ); }); }); const deleteButtons = this.shadowRoot!.querySelectorAll('.delete-btn'); deleteButtons.forEach(btn => { btn.addEventListener('click', () => { const id = parseInt(btn.getAttribute('data-id')!); this.dispatchEvent( new CustomEvent('delete-item', { detail: { id }, composed: true, bubbles: true, }), ); }); }); } update(): void { if (!this.shadowRoot) { this.attachShadow({ mode: 'open' }); } this.shadowRoot!.innerHTML = this.render(); this.addEventListeners(); } connectedCallback(): void { this.attachShadow({ mode: 'open' }); this.shadowRoot!.innerHTML = this.render(); this.addEventListeners(); } } customElements.define('items-list', ItemsList);

**src\scripts\components\admin\model.ts**

import { updateCertificate, deleteCertificate, getCertificates, postCertificate, updateBook, deleteBook, getAuthors, getGenres, getBooks, postBook, } from '../../api/api'; import { Certificate, Author, Genre, Book } from '../../../types'; export class AdminModel extends EventTarget { set currentCategory(category: 'certificates' | 'books') { this.\_currentCategory = category; this.\_selectedId = null; this.dispatchEvent(new Event('update')); } get currentItems(): (Certificate | Book)[] { return this.\_currentCategory === 'books' ? this.\_books : this.\_certificates; } get selectedItem(): Certificate | undefined | Book { return this.currentItems.find(item => item.id === this.\_selectedId); } set selectedId(id: number | null) { this.\_selectedId = id; this.dispatchEvent(new Event('update')); } get currentCategory(): 'certificates' | 'books' { return this.\_currentCategory; } get authors(): Author[] { return this.\_authors; } get genres(): Genre[] { return this.\_genres; } private \_optimisticState: { certificates: Certificate[]; books: Book[]; } | null = null; private \_currentCategory: 'certificates' | 'books' = 'books'; private \_certificates: Certificate[] = []; private \_selectedId: number | null = null; private \_authors: Author[] = []; private \_genres: Genre[] = []; private \_books: Book[] = []; updateItemOptimistic( item: FormData, category: 'certificates' | 'books', ): void { this.\_optimisticState = { certificates: [...this.\_certificates], books: [...this.\_books], }; if (category === 'books') { const id = parseInt(item.get('id') as string); const index = this.\_books.findIndex(b => b.id === id); if (index !== -1) { const authorIds = (item.get('authorIds') as string) ?.split(',') .map(Number) .filter(id => !isNaN(id)) || this.\_books[index].authors.map(a => a.id); const genreIds = (item.get('genreIds') as string) ?.split(',') .map(Number) .filter(id => !isNaN(id)) || this.\_books[index].genres.map(g => g.id); this.\_books[index] = { ...this.\_books[index], cover: item.get('poster') ? `/images/covers/${(item.get('poster') as File).name}` : this.\_books[index].cover, path: item.get('file') ? `/books/${(item.get('file') as File).name}` : this.\_books[index].path, authors: this.\_authors.filter(a => authorIds.includes(a.id)), genres: this.\_genres.filter(g => genreIds.includes(g.id)), name: item.get('name') as string, }; } } else { const id = parseInt(item.get('id') as string); const index = this.\_certificates.findIndex(c => c.id === id); if (index !== -1) { this.\_certificates[index] = { ...this.\_certificates[index], img: item.get('img') ? `/images/certificates/${(item.get('img') as File).name}` : this.\_certificates[index].img, name: item.get('name') as string, text: item.get('text') as string, }; } } this.dispatchEvent(new Event('update')); } addItemOptimistic(item: FormData, category: 'certificates' | 'books'): void { this.\_optimisticState = { certificates: [...this.\_certificates], books: [...this.\_books], }; if (category === 'books') { const authorIds = (item.get('authorIds') as string) ?.split(',') .map(Number) .filter(id => !isNaN(id)) || []; const genreIds = (item.get('genreIds') as string) ?.split(',') .map(Number) .filter(id => !isNaN(id)) || []; const newBook: Book = { cover: item.get('poster') ? `/images/covers/${(item.get('poster') as File).name}` : '', path: item.get('file') ? `/books/${(item.get('file') as File).name}` : '', authors: this.\_authors.filter(a => authorIds.includes(a.id)), genres: this.\_genres.filter(g => genreIds.includes(g.id)), name: item.get('name') as string, id: Date.now(), }; this.\_books.push(newBook); } else { const newCert: Certificate = { img: item.get('img') ? `/images/certificates/${(item.get('img') as File).name}` : '', name: item.get('name') as string, text: item.get('text') as string, id: Date.now(), }; this.\_certificates.push(newCert); } this.dispatchEvent(new Event('update')); } async updateItem( item: FormData, category: 'certificates' | 'books', ): Promise<void> { if (category === 'books') { const updatedBook = await updateBook(item); const index = this.\_books.findIndex( b => b.id === parseInt(item.get('id') as string), ); if (index !== -1) { this.\_books[index] = updatedBook; } } else { const updatedCert = await updateCertificate(item); const index = this.\_certificates.findIndex( c => c.id === parseInt(item.get('id') as string), ); if (index !== -1) { this.\_certificates[index] = updatedCert; } } this.dispatchEvent(new Event('update')); } deleteItemOptimistic(id: number, category: 'certificates' | 'books'): void { this.\_optimisticState = { certificates: [...this.\_certificates], books: [...this.\_books], }; if (category === 'books') { this.\_books = this.\_books.filter(b => b.id !== id); } else { this.\_certificates = this.\_certificates.filter(c => c.id !== id); } if (this.\_selectedId === id) { this.\_selectedId = null; } this.dispatchEvent(new Event('update')); } async deleteItem( id: number, category: 'certificates' | 'books', ): Promise<void> { if (category === 'books') { await deleteBook(id.toString()); this.\_books = this.\_books.filter(b => b.id !== id); } else { await deleteCertificate(id.toString()); this.\_certificates = this.\_certificates.filter(c => c.id !== id); } if (this.\_selectedId === id) { this.\_selectedId = null; } this.dispatchEvent(new Event('update')); } async addItem( item: FormData, category: 'certificates' | 'books', ): Promise<void> { if (category === 'books') { const newBook = await postBook(item); this.\_books.push(newBook); } else { const newCert = await postCertificate(item); this.\_certificates.push(newCert); } this.dispatchEvent(new Event('update')); } rollbackOptimistic(category: 'certificates' | 'books'): void { if (this.\_optimisticState) { this.\_books = [...this.\_optimisticState.books]; this.\_certificates = [...this.\_optimisticState.certificates]; this.\_optimisticState = null; this.dispatchEvent(new Event('update')); } } async fetchAll(): Promise<void> { this.\_books = await getBooks(); this.\_certificates = await getCertificates(); this.\_genres = await getGenres(); this.\_authors = await getAuthors(); this.dispatchEvent(new Event('update')); } }

**src\scripts\components\admin\view.ts**

import { Certificate, Book } from '../../../types'; import { AdminModel } from './model'; import { ItemsList } from './list'; export class AdminView { private \_onSubmit: (data: FormData, isUpdate: boolean) => void; private \_onCategoryChange: (category: string) => void; private \_onDelete: (id: number) => void; private \_onSelect: (id: number) => void; private \_formContainer: HTMLDivElement; private \_root: HTMLElement; private \_model: AdminModel; private \_list: ItemsList; constructor( root: HTMLElement, model: AdminModel, onCategoryChange: (category: string) => void, onDelete: (id: number) => void, onSelect: (id: number) => void, onSubmit: (data: FormData, isUpdate: boolean) => void, ) { this.\_root = root; this.\_model = model; this.\_onCategoryChange = onCategoryChange; this.\_onDelete = onDelete; this.\_onSelect = onSelect; this.\_onSubmit = onSubmit; this.\_list = new ItemsList(); } render(): void { this.\_list.items = this.\_model.currentItems; this.\_root.innerHTML = ` <div class="container-fluid w-100 h-100 flex-grow-1"> <div class="row"> <div class="col-3"> <select class="form-select mb-3" id="category"> <option value="books" ${this.\_model.currentCategory === 'books' ? 'selected' : ''}>Книги</option> <option value="certificates" ${this.\_model.currentCategory === 'certificates' ? 'selected' : ''}>Сертификаты</option> </select> <div id="list-container"></div> <button class="btn btn-primary mt-3" id="addBtn">Создать</button> </div> <div class="col-9"> <div id="formContainer"></div> </div> </div> </div> `; const listContainer = this.\_root.querySelector( '#list-container', ) as HTMLDivElement; listContainer.appendChild(this.\_list); this.\_formContainer = this.\_root.querySelector( '#formContainer', ) as HTMLDivElement; this.addEventListeners(); } update(): void { console.log('Обновление списка, элементы:', this.\_model.currentItems); this.\_list.items = this.\_model.currentItems; this.\_list.update(); const { selectedItem } = this.\_model; console.log('Выбранный элемент:', selectedItem); this.renderForm(selectedItem); } private renderForm(item?: Certificate | Book): void { const category = this.\_model.currentCategory; this.\_formContainer.innerHTML = ` <form class="mt-4"> <input type="hidden" name="id" id="id" value="${item?.id || ''}" /> ${ category === 'books' ? ` <div class="mb-3"> <label for="name" class="form-label">Название</label> <input type="text" class="form-control" name="name" id="name" value="${(item as Book)?.name || ''}" required /> </div> <div class="mb-3"> <label class="form-label">Авторы</label> <div> ${this.\_model.authors .map( author => ` <div class="form-check"> <input class="form-check-input" type="checkbox" name="authorIds" id="author-${author.id}" value="${author.id}" ${item && (item as Book)?.authors?.some(a => a.id === author.id) ? 'checked' : ''}> <label class="form-check-label" for="author-${author.id}">${author.name}</label> </div> `, ) .join('')} </div> ${ item && (item as Book)?.authors?.length ? ` <div class="mt-2"> <small>Выбрано: ${(item as Book).authors.map(a => a.name).join(', ')}</small> </div> ` : '' } </div> <div class="mb-3"> <label class="form-label">Жанры</label> <div> ${this.\_model.genres .map( genre => ` <div class="form-check"> <input class="form-check-input" type="checkbox" name="genreIds" id="genre-${genre.id}" value="${genre.id}" ${item && (item as Book)?.genres?.some(g => g.id === genre.id) ? 'checked' : ''}> <label class="form-check-label" for="genre-${genre.id}">${genre.name}</label> </div> `, ) .join('')} </div> ${ item && (item as Book)?.genres?.length ? ` <div class="mt-2"> <small>Выбрано: ${(item as Book).genres.map(g => g.name).join(', ')}</small> </div> ` : '' } </div> <div class="mb-3"> <label for="poster" class="form-label">Обложка (PNG/JPEG)</label> <input type="file" class="form-control" name="poster" id="poster" accept="image/png,image/jpeg" /> ${item && (item as Book)?.cover ? `<div class="mt-2"><small>Текущий файл: ${(item as Book)?.cover}</small></div>` : ''} </div> <div class="mb-3"> <label for="file" class="form-label">Файл книги (EPUB)</label> <input type="file" class="form-control" name="file" id="file" accept=".epub" /> ${item && (item as Book)?.path ? `<div class="mt-2"><small>Текущий файл: ${(item as Book)?.path}</small></div>` : ''} </div> ` : ` <div class="mb-3"> <label for="name" class="form-label">Название</label> <input type="text" class="form-control" name="name" id="name" value="${(item as Certificate)?.name || ''}" required /> </div> <div class="mb-3"> <label for="text" class="form-label">Текст</label> <input type="text" class="form-control" name="text" id="text" value="${(item as Certificate)?.text || ''}" required /> </div> <div class="mb-3"> <label for="img" class="form-label">Изображение (PNG/JPEG)</label> <input type="file" class="form-control" name="img" id="img" accept="image/png,image/jpeg" /> ${item && (item as Certificate)?.img ? `<div class="mt-2"><small>Текущий файл: ${(item as Certificate)?.img}</small></div>` : ''} </div> ` } <button type="submit" class="btn btn-primary">${item ? 'Сохранить' : 'Добавить'}</button> </form> `; const form = this.\_formContainer.querySelector('form') as HTMLFormElement; form.addEventListener('submit', e => { try { e.preventDefault(); if (!form.checkValidity()) { form.reportValidity(); return; } const formData = new FormData(form); const authorIds = Array.from( form.querySelectorAll('input[name="authorIds"]:checked'), ).map(input => (input as HTMLInputElement).value); const genreIds = Array.from( form.querySelectorAll('input[name="genreIds"]:checked'), ).map(input => (input as HTMLInputElement).value); if (authorIds.length) { formData.set('authorIds', authorIds.join(',')); } if (genreIds.length) { formData.set('genreIds', genreIds.join(',')); } console.log('LOG: authorIds:', formData.get('authorIds')); console.log('LOG: genreIds:', formData.get('genreIds')); const isUpdate = !!item; this.\_onSubmit(formData, isUpdate); } catch (err) { console.error('Ошибка при обработке формы:', err); } }); } private addEventListeners(): void { const categorySelect = this.\_root.querySelector( '#category', ) as HTMLSelectElement; categorySelect.addEventListener('change', () => { this.\_onCategoryChange(categorySelect.value); }); const addBtn = this.\_root.querySelector('#addBtn') as HTMLButtonElement; addBtn.addEventListener('click', () => { this.\_model.selectedId = null; this.renderForm(); }); this.\_list.addEventListener('active-changed', (e: Event) => { const { active } = (e as CustomEvent).detail; const item = this.\_list.items[active]; this.\_onSelect(item.id); }); this.\_list.addEventListener('delete-item', (e: Event) => { const { id } = (e as CustomEvent).detail; this.\_onDelete(id); }); } }

**src\scripts\components\book-form.ts**

import styles from '../../styles/style.scss?inline'; import { useDumbState } from '../utils'; type BookFormT = 'edit' | 'add'; export class BookForm extends HTMLElement { static get observedAttributes() { return ['type', 'id']; } private \_id: number | null = null; private \_type: BookFormT = 'add'; constructor() { super(); } attributeChangedCallback( name: string, oldValue: string, newValue: BookFormT, ) { if (name === 'type') { this.\_type = newValue; } if (name === 'id') { this.\_id = +newValue; } } connectedCallback() { this.attachShadow({ mode: 'open' }); this.shadowRoot!.innerHTML = this.render(); this.addSubmitListener(); } public updateData() { this.shadowRoot!.innerHTML = this.render(); } private render() { const [data, setData] = useDumbState<BookFormT>('add', () => {}); return ` <style> ${styles} </style> <section class="container"> <div class="row align-items-center mt-0 mt-md-4 mt-lg-4"> <h1 class="text-center fs-2 mb-0">${this.\_type === 'add' ? 'Добавление' : 'Редактирование'} книги:</h1> </div> </section> <form id="bookForm" class="container mt-0 mt-md-4 mt-lg-4 max-width-500" > <div class="mb-0 mb-md-2 mb-lg-4"> <label for="genre" class="form-label">Жанр</label> <select class="form-select" id="genre" name="genre" required> <option value="" selected disabled>Выберите жанр</option> <option value="Фэнтези">Фэнтези</option> <option value="Научная фантастика">Научная фантастика</option> <option value="Детектив">Детектив</option> <option value="Роман">Роман</option> <option value="Ужасы">Ужасы</option> <option value="Триллер">Триллер</option> <option value="Классика">Классика</option> <option value="Другое">Другое</option> </select> </div> <div class="mb-0 mb-md-2 mb-lg-4"> <label for="title" class="form-label">Название книги</label> <input type="text" class="form-control" id="title" name="title" placeholder="Название книги" /> </div> <div class="mb-0 mb-md-2 mb-lg-4"> <label for="author" class="form-label">Автор</label> <input type="text" class="form-control" id="author" name="author" placeholder="Автор" /> </div> <div class="mb-0 mb-md-2 mb-lg-4"> <label for="poster">Обложка (png, jpg)</label> <input type="file" class="form-control" id="poster" name="poster" accept=".jpg, .jpeg, .png" required /> </div> <div class="mb-2 mb-md-4 mb-lg-6"> <label for="file" class="form-label">Файл книги (EPUB)</label> <input type="file" class="form-control" name="file" id="file" accept=".epub" required /> </div> <div class="d-flex justify-content-center"> <button type="submit" class="btn btn-primary"> ${this.\_type === 'add' ? 'Загрузить' : 'Сохранить'} книгу </button> </div> </form> `; } private async handleSubmit(event: Event) { event.preventDefault(); const form = event.target as HTMLFormElement; const formData = new FormData(form); const url = `http://localhost:3000${ this.\_type === 'add' ? `/add-book` : `/edit-book/${this.\_id ?? ''}` }`; const response = await fetch(url, { method: 'POST', body: formData, }); if (response.ok) { alert(`Книга успешно ${this.\_type === 'add' ? 'добавлена' : 'изменена'}`); form.reset(); } else { alert( `Ошибка при ${this.\_type === 'add' ? 'добавлении' : 'изменении'} книги`, ); } } private addSubmitListener() { const form = this.shadowRoot!.querySelector('form'); form?.addEventListener('submit', this.handleSubmit); } }

**src\scripts\components\card.ts**

import stylesString from './card.scss?inline'; export class Card extends HTMLElement { static get observedAttributes() { return ['img', 'name', 'link', 'icon']; } /\* захардкоженная иконка на кнопке \*/ private \_icon: string = '/images/icons/add.svg'; private \_link: string = ''; private \_name: string = ''; /\* обложка \*/ private \_img: string = ''; constructor() { super(); } private render() { if (this.\_link !== '/') { this.\_icon = '/images/icons/resume.svg'; } return ` <style> ${stylesString} </style> <div class="card"> <div class="card\_\_cover"> <img class="card\_\_cover\_\_img" src="${ this.\_img }" alt="${this.\_name} cover" lazy /> </div> <div class="card\_\_description"> <span class="card\_\_description\_\_text"> ${this.\_name} </span> <button class="card\_\_button"> <a href="${this.\_link}"> <img src="${this.\_icon}" class="card\_\_button\_\_img" alt="add icon" /> </a> </button> </div> </div> `; } attributeChangedCallback(name: string, oldValue: string, newValue: string) { if (name === 'img') { this.\_img = newValue; } else if (name === 'href') { this.\_link = newValue; } else if (name === 'name') { this.\_name = newValue; } else if (name === 'icon') { if (newValue) { this.\_icon = newValue; } } } updateData() { this.\_img = this.getAttribute('img') ?? ''; this.\_link = this.getAttribute('link') ?? ''; this.\_name = this.getAttribute('name') ?? ''; this.\_icon = this.getAttribute('icon') ?? '/images/icons/add.svg'; } connectedCallback() { this.attachShadow({ mode: 'open' }); this.updateData(); this.shadowRoot!.innerHTML = this.render(); } }

**src\scripts\index.ts**

// eslint-disable-next-line @typescript-eslint/no-unused-vars import \* as bootstrap from 'bootstrap'; import 'bootstrap-icons/font/bootstrap-icons.css'; console.log(bootstrap); import { Card } from './components/card'; customElements.define('product-card', Card);

**src\scripts\utils.ts**

export function useDumbState<T>( initialValue: T, render: () => void, ): [T, (newValue: T) => void] { let value = initialValue; const getValue = (): T => value; const setValue = (newValue: T) => { value = newValue; render(); }; return [getValue(), setValue]; }

**src\sertificates.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <link rel="icon" type="image/svg" href="/icon.svg" /> <title>Сертификаты</title> <link rel="stylesheet" href="../styles/style.scss" /> <script src="../scripts/certificates/certificatesFetcher.ts" type="module" ></script> <script src="../scripts/index.ts" type="module"></script> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li><a class="dropdown-item" href="/user">user</a></li> <li> <a class="dropdown-item" href="/sertificates">Сертификаты</a> </li> <li><hr class="dropdown-divider" /></li> <li> <a class="dropdown-item" href="/logout">Выйти</a> </li> </ul> </nav> </section> </header> <main class="d-flex flex-column gap-4 justify-content-center"> <div class="d-flex justify-content-center p-4"> <div class="d-flex align-items-center justify-content-between border rounded p-5" > <h2 class="text-center">Дарите знания!</h2> <div class="overflow-hidden"> <img class="w-100" src="/images/posters/book-present.svg" alt="banner logo" /> </div> </div> </div> <div id="certificates-list" class="d-grid grid-cols-1 grid-md-cols-2 gap-4 p-4" ></div> </main> <footer class="footer"> <div class="footer\_\_img"> <img class="" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="d-flex flex-column flex-md-row flex-lg-row gap-2"> <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**src\types.d.ts**

export interface Book { authors: Author[]; genres: Genre[]; cover: string; name: string; path: string; id: number; } export interface Certificate { name: string; text: string; img: string; id: number; } export interface Author { name: string; id: number; } export interface Genre { name: string; id: number; }

**src\user.html**

<!doctype html> <html lang="ru"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" /> <title>Личный кабинет</title> <link rel="icon" type="image/svg" href="/icon.svg" /> <link rel="stylesheet" href="../styles/style.scss" /> <script type="module" src="../scripts/index.ts"></script> </head> <body> <header class="border-bottom d-flex justify-content-between"> <a href="/" class=""> <img src="/icon.svg" alt="Elib logo" /> </a> <section class="flex-grow-1 d-flex justify-content-around justify-content-md-around justify-content-lg-between align-items-center gap-2 px-2" > <form class="flex-grow-1 d-flex max-width-500 align-items-center gap-2"> <input name="search" class="flex-grow-1 form-input" type="text" placeholder="Поиск" spellcheck /> <button type="submit" class="btn-secondary"> <img src="/images/icons/search.svg" alt="search icon" /> </button> </form> <nav class="dropdown"> <div class="btn-secondary" type="button" id="dropdownMenuButton1" data-bs-toggle="dropdown" data-bs-auto-close="true" aria-expanded="false" > <img src="/images/icons/account.svg" alt="user icon" /> </div> <ul class="dropdown-menu dropdown-menu-end dropdown-menu-lg" aria-labelledby="dropdownMenuButton1" data-bs-popper="static" > <li><a class="dropdown-item" href="/user">user</a></li> <li> <a class="dropdown-item" href="/sertificates">Сертификаты</a> </li> <li><hr class="dropdown-divider" /></li> <li> <a class="dropdown-item" href="/logout">Выйти</a> </li> </ul> </nav> </section> </header> <main class="main-content"> <section> <div class="main-content\_\_heading-block"> <h2>Ваши книги, <span style="color: red">user</span></h2> <hr /> </div> <div class="main-content\_\_books"> <product-card img="/images/covers/harrypotter.jpg" link="/book.html?book=harry-potter" name="Гарри Поттер и философский камень" ></product-card> <product-card img="/images/covers/harrypotter.jpg" link="/book.html?book=harry-potter" name="Гарри Поттер и философский камень" ></product-card> </div> </section> </main> <footer class="footer"> <div class="footer\_\_img"> <img class="footer\_\_img--content" src="/icon.svg" alt="logo" /> </div> <div class="flex-grow-1 d-flex gap-4 justify-content-between"> <div class="footer\_\_links d-flex flex-column flex-md-row flex-lg-row gap-2" > <a class="link" href="/">Главная</a> <a class="link" href="/register">Регистрация</a> <a class="link" href="/sertificates">Купоны и бонусы</a> </div> <div>2025 &#128218;</div> </div> </footer> </body> </html>

**tsconfig.json**

{ "compilerOptions": { "target": "ES2020", "useDefineForClassFields": true, "module": "ESNext", "lib": ["ES2020", "DOM", "DOM.Iterable"], "skipLibCheck": true, "moduleResolution": "bundler", "allowImportingTsExtensions": true, "resolveJsonModule": true, "isolatedModules": true, "noEmit": true, "strict": true, "noUnusedLocals": true, "noUnusedParameters": true, "noFallthroughCasesInSwitch": true, "types": ["vite/client"] }, "include": ["src/\*\*/\*.ts", "src/\*\*/\*.js", "src/\*\*/\*.html", "src/scripts/\*\*/\*.ts", "server/\*\*/\*.ts", "src/types.d.ts", "server/index.ts"] }

**vite.config.js**

import { defineConfig } from 'vite'; import { resolve } from 'path'; export default defineConfig({ root: 'src', resolve: { alias: { '~bootstrap': resolve(\_\_dirname, 'node\_modules/bootstrap'), }, }, build: { outDir: '../dist', rollupOptions: { input: { index: 'index.html', book: 'book.html', login: 'login.html', register: 'register.html', sertificates: 'sertificates.html', user: 'user.html', admin: 'admin.html', }, output: { entryFileNames: 'assets/js/[name].js', chunkFileNames: 'assets/js/[name].js', assetFileNames: assetInfo => { if (/\.(css|scss|sass)$/.test(assetInfo.name)) { return 'assets/css/[name][extname]'; } return 'assets/[name][extname]'; }, }, }, }, });