

Реализация полнотекстового поиска по контенту сайта в веб-приложении на основе React и Node.js

Андриенко Иван Сергеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Пащенко Екатерина Александровна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается реализация системы полнотекстового поиска по основным разделам информационного веб-сайта, разработанного с использованием связки React на стороне клиента и Node.js с ORM Prisma на стороне сервера. Описывается архитектура поискового механизма, включающая форму поиска в шапке сайта, страницу отображения результатов и серверный API-эндпоинт, обрабатывающий запросы и выполняющий поиск по нескольким категориям данных: новостям, документам, объявлениям, образовательным программам и информации о сотрудниках. Приводится описание принципов формирования URL-запросов, обработки результатов на фронтенде, а также логики выбора целевых маршрутов при переходе к детальным страницам.

Ключевые слова: полнотекстовый поиск, веб-приложение, React, Node.js, Prisma, API, пользовательский интерфейс, новости, образовательные программы.

Implementation of full-text search for website content in a React-based web application and Node.js

Andrienko Ivan Sergeevich

Sholom-Aleichem Priamursky State University

Student

Paschenko Ekaterina Alexandrovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the implementation of a full-text search system for the main sections of an information website developed using a combination of React on the client side and Node.js with ORM Prisma on the server side. The architecture of the search engine is described, which includes a search form in the header of the site, a

results display page, and a server-side API endpoint that processes requests and performs searches across several categories of data: news, documents, ads, educational programs, and information about employees. The principles of URL query formation, processing of results on the frontend, as well as the logic of selecting target routes when navigating to detailed pages are described.

Keywords: full-text search, web application, React, Node.js, Prisma, API, user interface, news, educational programs.

1 Введение

1.1 Актуальность

Современные образовательные и информационные веб-ресурсы содержат значительные объёмы структурированных и неструктурированных данных, включая новости, нормативные документы, объявления, сведения об образовательных программах и персонале. При увеличении объёма информации простое меню навигации и статические разделы перестают быть достаточными для быстрого доступа к нужным материалам. Пользователи ожидают наличие единого поискового механизма, позволяющего по ключевым словам находить сведения независимо от того, к какому разделу сайта они относятся. Реализация такого поиска требует согласованной работы клиентской и серверной частей приложения, корректной обработки запросов, продуманной структуры базы данных и удобного интерфейса представления результатов. В условиях использования современных веб-технологий, таких как React на фронтенде и Node.js с ORM Prisma на бэкенде, задача проектирования централизованной поисковой системы приобретает особую актуальность для разработчиков сайтов образовательных организаций и информационных порталов.

1.2 Обзор исследований

Разработка веб-приложений на основе React и TypeScript активно рассматривается в научной литературе. В работе М.М. Ковцура, В.Е. Филимонова и Е.К. Щёголева анализируются подходы к созданию многопользовательских систем на базе React, где особое внимание уделяется организации типизации и архитектуры клиентской части [1]. А.А. Дудак в своём исследовании проводит сравнительный анализ инструментов разработки для проектных систем, выделяя преимущества стека технологий React и TypeScript по производительности и удобству сопровождения кода [2]. В статье Д.А. Хайруллина и М.Р. Богданова рассматриваются особенности применения статической типизации в React-приложениях, включая методики типизации компонентов, хуков и состояния, а также влияние строгой типизации на надёжность пользовательских интерфейсов [3]. К.Р. Удальцов исследует вопросы обеспечения качества кода в командной разработке, предлагая формализацию шаблонов безопасных pull request-ов для проектов на React и TypeScript в рамках Git Flow [4]. Особое внимание созданию библиотек компонентов на основе TypeScript уделяется в работе Т.М. Воронко, где описываются принципы модульности, повторного использования кода и

проектирования архитектуры компонентов для масштабируемых React-решений [5].

1.3 Цель исследования

Целью данной работы является разработка и описание механизма глобального поиска по контенту сайта, включающего новости, документы, объявления, образовательные программы и данные о сотрудниках, в рамках веб-приложения на основе React и Node.js. В рамках исследования рассматриваются организация поисковой формы в шапке сайта, реализация страницы результатов, построение серверного API для выполнения запросов с учётом минимальной длины поиска, регистронезависимого сопоставления строк и ограничений на количество возвращаемых записей, а также формирование удобного для пользователя представления найденной информации.

2 Материалы и методы

В качестве основы использовалось веб-приложение, реализованное на библиотеке React. Форма поиска в компоненте Header была реализована в виде HTML-формы с обработчиком события отправки, перехватывающим стандартное поведение браузера.

3 Результаты и обсуждения

Реализация глобального полнотекстового поиска потребовала согласованной работы клиентской и серверной частей приложения. Первым шагом стала разработка поисковой формы, расположенной в верхней панели хедера. Данный интерфейсный элемент обеспечивает быстрый доступ к поиску с любой страницы сайта и выполняет первичную обработку пользовательского запроса. HTML-форма содержит поле ввода текста и обработчик события onSubmit, который предотвращает стандартное поведение браузера, извлекает поисковую строку с помощью FormData и перенаправляет пользователя на страницу результатов с параметром q, добавленным в URL. Фрагмент реализации формы поиска представлен на рисунке (рис. 1).

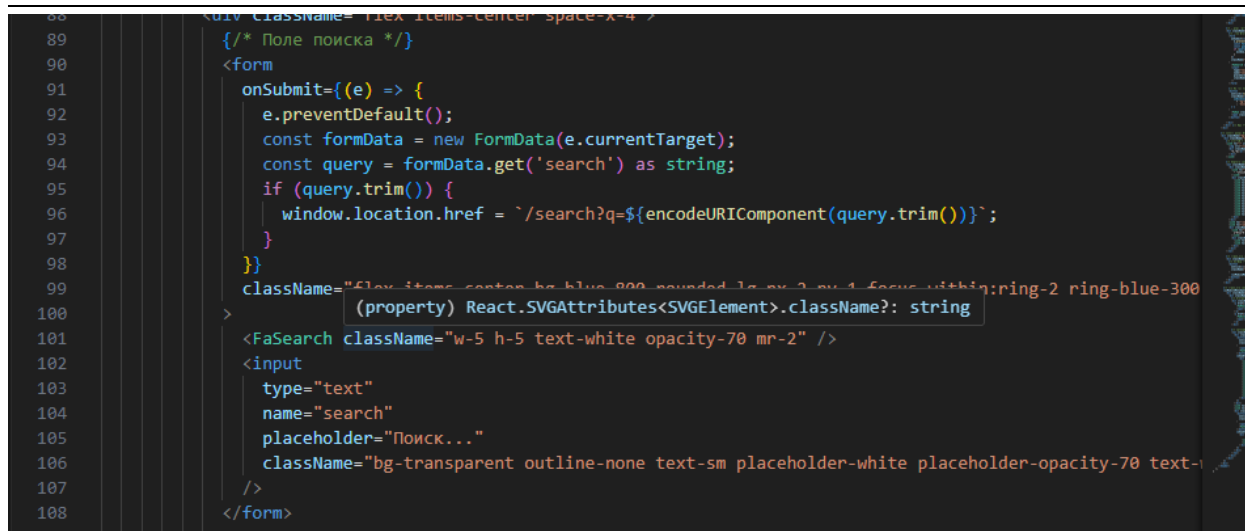


Рис. 1. Форма поиска, размещённая в хедере сайта, с обработкой события отправки

После выполнения перенаправления пользователь попадает на страницу SearchResults, где извлечение параметра *q* осуществляется из адресной строки. В компоненте используется эффект *useEffect* для отслеживания изменений поискового параметра. При наличии запроса запускается функция *performSearch*, выполняющая асинхронный запрос к API. На время выполнения запроса на экране отображается индикатор загрузки, что улучшает взаимодействие с интерфейсом и информирует пользователя о текущем процессе. Основная логика получения результатов поиска и их сохранения в состояние компонента показана на рисунке (рис. 2).

```
23 export default function SearchResults() {
24
25   const [results, setResults] = useState({
26     news: SearchResult[];
27     documents: SearchResult[];
28     announcements: SearchResult[];
29     educationPrograms: SearchResult[];
30     employees: SearchResult[];
31     structureDocuments: SearchResult[];
32     total: number;
33   });
34   const [loading, setLoading] = useState(true);
35
36   useEffect(() => {
37     if (query) {
38       performSearch(query);
39     }
40   }, [query]);
41
42   const performSearch = async (searchQuery: string) => {
43     try {
44       setLoading(true);
45       const response = await fetch(`/api/search?q=${encodeURIComponent(searchQuery)}`);
46       const data = await response.json();
47       setResults(data);
48     } catch (error) {
49       console.error('Search error:', error);
50     } finally {
51       setLoading(false);
52     }
53   };
54 }
```

Рис. 2. Асинхронный запрос к API поиска на клиентской стороне

Результаты поиска группируются по категориям данных, доступным в системе, включая новости, документы, сотрудников, объявления и образовательные программы. Для каждой категории выводится список найденных элементов, оформленный с использованием типографических классов и иконок, что обеспечивает удобство навигации и восприятия контента. Форматирование дат публикации выполняется с использованием локализации на русский язык, что делает интерфейс более привычным для пользователя. Каждый элемент результата является кликабельным и ведёт на соответствующую страницу раздела. Итоговый вид отображения результатов поиска на клиенте приведён на рисунке (рис. 3).

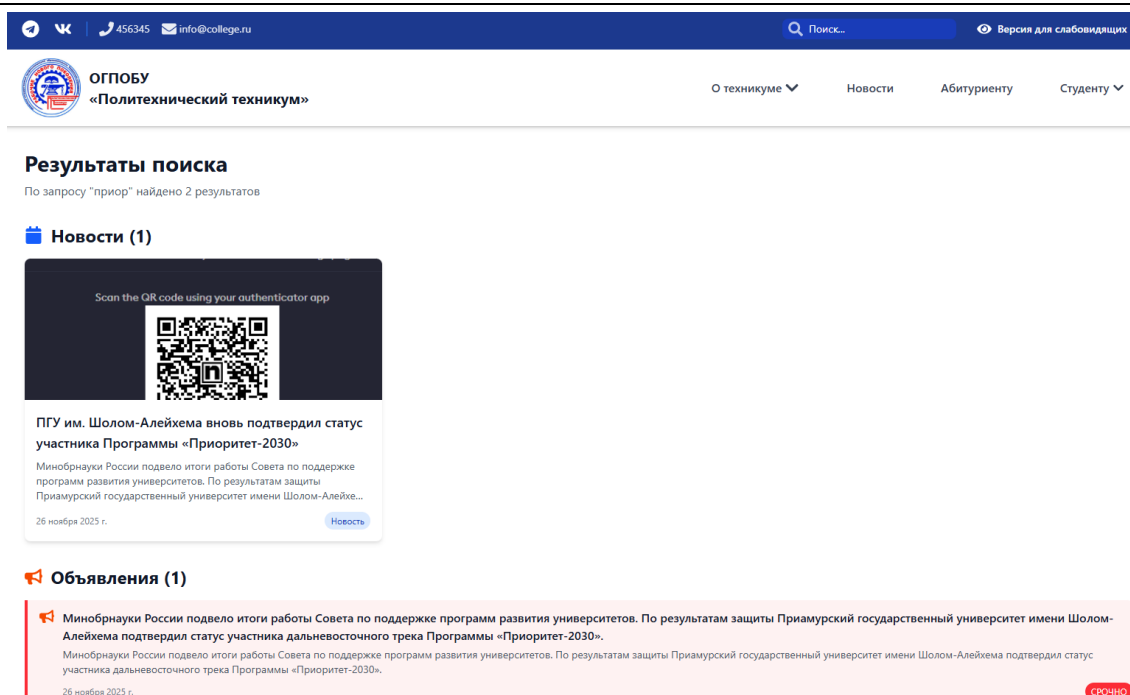


Рис. 3. Интерфейс страницы SearchResults с группировкой найденных материалов по категориям

Серверная часть поискового механизма реализована в файле `search.js`. Обработчик GET-запроса `/api/search` принимает параметры `q` и `limit`, после чего выполняет последовательность проверок и преобразований. Минимальная длина поискового запроса ограничена двумя символами, что предотвращает чрезмерное количество запросов к базе данных и бессмысленные поисковые операции. Входная строка нормализуется методом `trim` и приводится к нижнему регистру для обеспечения регистронезависимого поиска. Далее поочерёдно выполняются запросы к различным таблицам базы данных с использованием ORM `Prisma`. Каждая выборка включает несколько полей сущности, чтобы повышать полноту поиска: заголовки, описания, тексты новостей, ФИО и должности сотрудников, содержание объявлений и образовательных программ. Обработка поисковых условий выполняется оператором `contains` с параметром `mode: 'insensitive'`. Фрагмент серверной логики представлен на рисунке (рис. 4, 5).

```
1 import express from 'express';
2 import { PrismaClient } from '@prisma/client';
3
4 const prisma = new PrismaClient();
5 const router = express.Router();
6
7 // Поиск по всему контенту
8 router.get('/search', async (req, res) => {
9   try {
10     const { q: query, limit = 20 } = req.query;
11
12     if (!query || query.trim().length < 2) {
13       return res.json({
14         news: [],
15         documents: [],
16         announcements: [],
17         total: 0
18       });
19     }
20
21     const searchTerm = query.trim().toLowerCase();
22     const limitNum = parseInt(limit);
23
24     // Поиск по новостям
25     const news = await prisma.news.findMany({
26       where: {
27         OR: [
28           { title: { contains: searchTerm, mode: 'insensitive' } },
29           { fullText: { contains: searchTerm, mode: 'insensitive' } },
30           { shortDescription: { contains: searchTerm, mode: 'insensitive' } }
31         ]
32       },
33       select: {
34         id: true,
35         title: true,
36         slug: true,
37         shortDescription: true,
38         publishedAt: true,
39         previewImage: true
40       }
41     });
42   } catch (error) {
43     console.error('Error in search handler:', error);
44     return res.status(500).json({ error: 'Internal server error' });
45   }
46   return res.json({ news });
47 });
```

Рис. 4. Серверный код обработки поискового запроса с использованием Prisma

```
125 // Поиск по образовательным программам
126 const educationPrograms = await prisma.educationProgram.findMany({
127   where: {
128     OR: [
129       { name: { contains: searchTerm, mode: 'insensitive' } },
130       { program: { contains: searchTerm, mode: 'insensitive' } },
131       { subjects: { contains: searchTerm, mode: 'insensitive' } }
132     ]
133   },
134   select: {
135     id: true,
136     name: true,
137     program: true,
138     code: true
139   },
140   take: limitNum
141 });
142
143 res.json({
144   news: news.map(item => ({ ...item, title: item.title, type: 'news' })),
145   documents: documents.map(item => ({ ...item, title: item.field, description: item.fileName, fileUrl: it
146   announcements: announcements.map(item => ({ ...item, urgent: item.isUrgent, type: 'announcement' })),
147   educationPrograms: educationPrograms.map(item => ({ ...item, title: item.name, description: item.progra
148   employees: employees.map(item => ({ ...item, title: item.fio, description: `${item.post} - ${item.name}
149   structureDocuments: structureDocuments.map(item => ({ ...item, title: item.title, description: item.des
150   total: news.length + documents.length + announcements.length + educationPrograms.length + employees.len
151 });
152
153 } catch (error) {
154   console.error('Search error:', error);
155   res.status(500).json({ error: 'Ошибка поиска' });
156 }
157 });
158
159 export default router;
```

Рис. 5. Серверный код обработки поискового запроса с использованием Prisma

Каждый набор данных сортируется по дате публикации в порядке убывания, что позволяет отображать наиболее актуальные результаты в начале списка. Также применяется ограничение на количество элементов, задаваемое параметром `limit`, что способствует контролю нагрузки на сервер и ускоряет процесс получения данных. По завершении выборки формируется единый объект, содержащий массивы найденных данных для всех категорий и итоговое количество обнаруженных материалов. Этот объект возвращается клиенту в формате JSON, где он используется для построения интерфейса результатов.

4 Выводы

Проведённая разработка показала, что интеграция единого механизма полнотекстового поиска в веб-приложение на основе React и Node.js позволяет обеспечить быстрый доступ к разнородным данным сайта и повысить удобство его использования. Реализованная система объединяет обработку поискового запроса на клиентской стороне, асинхронное взаимодействие с серверным API и полнотекстовый поиск в базе данных через Prisma, что обеспечивает корректную работу поиска по новостям, документам, объявлениям, образовательным программам и данным о сотрудниках. Тестирование подтвердило устойчивость работы, предсказуемость поведения

интерфейса и корректность поиска даже при высокой вариативности пользовательских запросов, что подтверждает эффективность предложенного решения.

Библиографический список

1. Ковцур М.М., Филимонов В.Е., Щёголев Е.К. Исследование подходов для реализации многопользовательского React-приложения на языке TypeScript // Мобильный бизнес: перспективы развития и реализации систем радиосвязи в России и за рубежом. Материалы 53-й Международной конференции. Москва, 2024. С. 58–61.
2. Дудак А.А. Сравнительный анализ инструментов разработки для систем управления проектами: преимущества стек-технологий TypeScript и React // Новая наука: от идеи к результату. 2024. № 9. С. 32–40.
3. Хайруллин Д.А., Богданов М.Р. Изучение использования TypeScript в React-приложениях, преимущества статической типизации, подходы к типизации компонентов, хуков и состояния // Международный журнал гуманитарных и естественных наук. 2024. № 12-3 (99). С. 233–238.
4. Удальцов К.Р. Формализация шаблонов безопасных pull request-ов для React-приложений с TypeScript в командах с Git Flow // Актуальные проблемы социально-гуманитарного и научно-технического знания. 2025. № 4 (44). С. 35–36.
5. Воронко Т.М. Создание React-библиотеки с использованием TypeScript // Электронные системы и технологии. Материалы 58-й научной конференции аспирантов, магистрантов и студентов БГУИР. Минск, 2022. С. 200–202.