МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Костромской государственный университет» (КГУ)

Институт “Высшая ИТ школа”

Кафедра информационных систем и технологий

Направление подготовки/Специальность* 09.03.02

Информационные системы и технологии

Профиль Разработка программного обеспечения информационных систем

Дисциплина Методы и средства проектирования информационных систем

ОТЧЕТ ПО МЕЖДИСЦИПЛИНАРНОМУ ПРОЕКТУ

Проектирование и разработка модуля "база знаний" для сервиса настольных ролевых игр "ROLE FORGE"

Направление подготовки: 09.03.02

Информационные системы и технологии

Форма обучения очная

Выполнили студент Коморин Алексей Романович

Курс 3 Группа 22-ИСбо-2б

Проверил _____ (ученая степень, ученое звание)

_____ (фамилия, имя, отчество)

Оценка преподавателя_____

Подпись преподавателя _____

Кострома

2025

Оглавление

1. ПОСТАНОВКА ЗАДАЧИ

Современные приложения и сайты имеют довольно сложный функционал, с которым не каждый пользователь захочет разбираться. Чтобы пользователь мог разобраться во всех необходимых ему функциях, создаются различные «Wiki» или же «базы знаний». Благодаря ним пользователь сможет самостоятельно разобраться в интересующих его вопросах.

2. ЦЕЛЬ РАБОТЫ

Проектирование и разработка модуля «База знаний» (Далее БЗ) для:

- Обеспечения **логичной навигации и иерархии информации**;
- Обеспечения пользователю **поиска нужных данных**;
- Создания **гибкой архитектуры**, которую можно масштабировать (добавлять категории, статьи, типы материалов);
- Поддержки **разного типа контента**: текст, изображения, вложения, ссылки, видео и т.д.;

3. ЗАДАЧИ РАБОТЫ

Для реализации веб-приложения, представляющего собой интерактивную базу знаний, необходимо выполнить комплекс задач. На этапе проектирования особое внимание уделяется выбору и организации структуры данных, обеспечивающей эффективное хранение и доступ к информации. С точки зрения клиентского интерфейса важно создать удобную, интуитивно понятную навигацию, а также обеспечить адаптивный и современный внешний вид. Таким образом задачи можно разделить на 2 блока:

Проектировка БД

1. Выбрать подходящую СУБД для хранения и обработки данных
2. Определить основные категории и типы информации, которые будут храниться в базе знаний
3. Создать ER-диаграмму для визуализации сущностей и связей между ними
4. Реализовать структуру базы данных на выбранной СУБД

Frontend (клиентская часть)

1. Разработать главную страницу с отображением каталогов и категорий базы знаний
2. Реализовать страницу просмотра статей с навигацией и отображением содержимого
3. Изучить и применить архитектурный подход Feature-Sliced Design
4. Создать пользовательские стили с использованием SCSS-переменных
5. Сверстать основные компоненты интерфейса в соответствии с дизайном

4. СРАВНИТЕЛЬНЫЙ АНАЛИЗ АНАЛОГОВ

Для того чтобы определить структуру и содержание собственной базы знаний по настольным ролевым играм, необходимо провести анализ существующих аналогов. Изучение таких ресурсов, как Roll20 Compendium, Pathfinder Wiki и Paizo PRD, позволяет понять, какие типы информации наиболее востребованы среди игроков и мастеров, какие функции обеспечивают удобство использования, а также какие подходы к организации данных оказываются наиболее эффективными. Сравнение этих платформ помогает выявить ключевые элементы, которые стоит включить в проект, чтобы обеспечить его актуальность, полноту и практическую ценность для целевой аудитории.

Roll20 Compendium — это универсальная база знаний для настольных ролевых игр, доступная в любом месте и в любое время. С её помощью игроки и мастера могут быстро находить правила, заклинания, монстров, предметы и другие игровые элементы, синхронизируя всю необходимую информацию в одном удобном месте. Roll20 Compendium помогает преодолевать хаос игровых сессий, устанавливать приоритеты в изучении правил, избегать перегрузок от поиска информации и достигать баланса между подготовкой к игре и самим игровым процессом. Благодаря интеграции с игровыми столами Roll20 и поддержке множества игровых систем, таких как Dungeons & Dragons, Pathfinder и Call of Cthulhu, этот инструмент становится незаменимым для тех, кто стремится к увлекательным и организованным игровым сессиям.

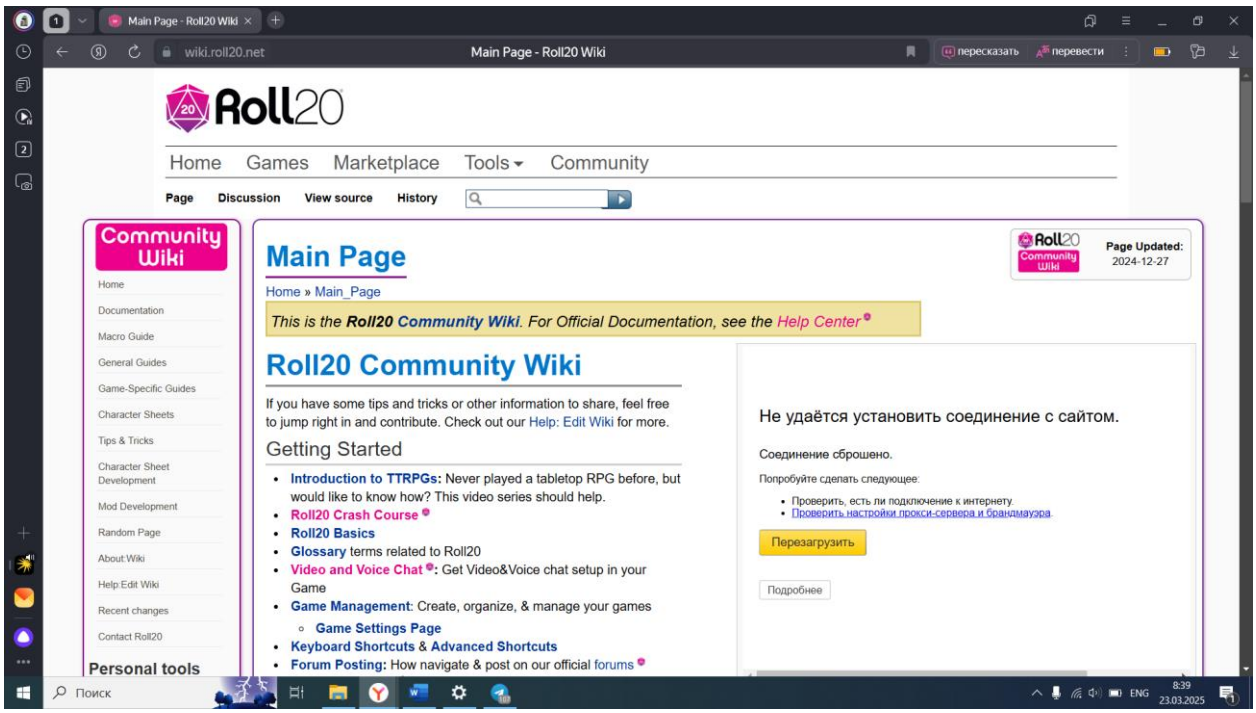

Рис. 1. Roll20 Compendium

Pathfinder Wiki на платформе Fandom — это специализированная база знаний, доступная в любое время и в любом месте. Она создана для игроков и мастеров, чтобы они могли глубже погружаться в детали сеттинга, изучать правила, создавать персонажей и планировать приключения — объединяя всю необходимую информацию в одном удобном пространстве. Она помогает справляться со сложными игровыми механиками, расставлять приоритеты в изучении мира, избегать путаницы в правилах и находить баланс между вдохновением и технической подготовкой.

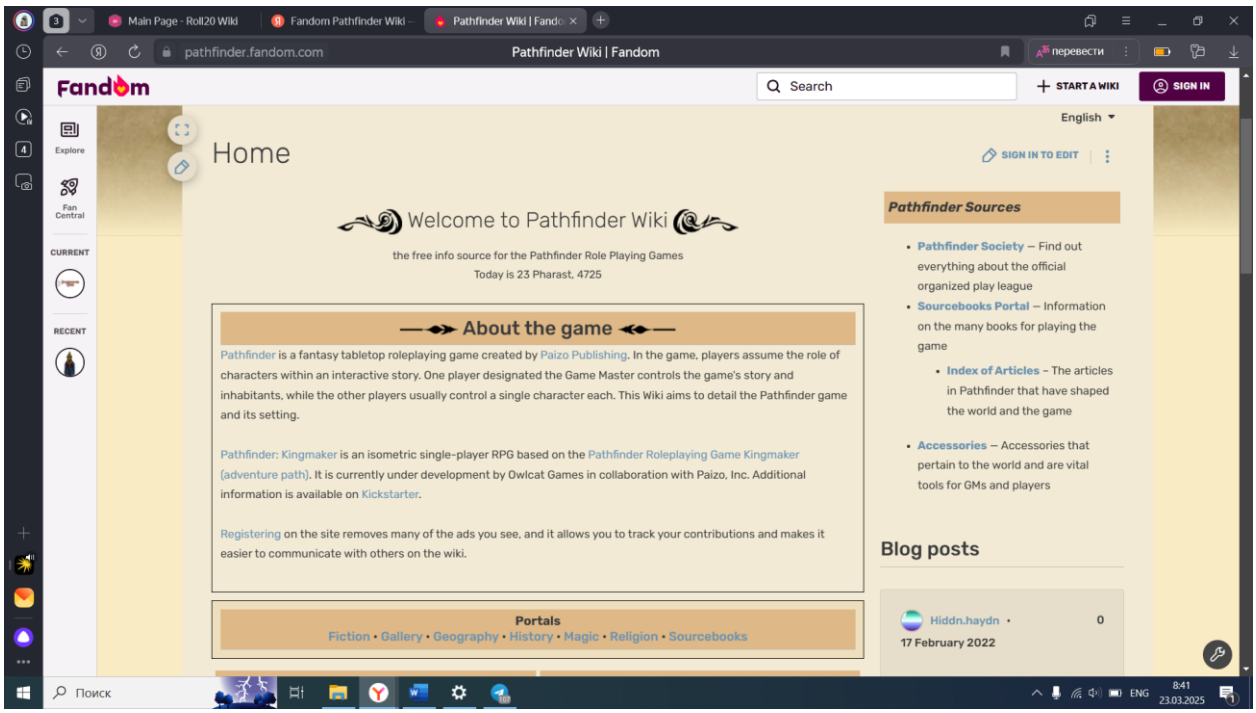

Рис. 2. Pathfinder

Paizo.com/PRD — это официальная база знаний по Pathfinder, доступная в любом месте и в любое время. С её помощью игроки и мастера могут быстро находить правила, классы, расы, заклинания, монстров и другие элементы игры, синхронизируя всю необходимую информацию в одном удобном месте. PRD помогает преодолевать сложности игровых систем, устанавливать приоритеты в создании персонажей, избегать перегрузок от изучения правил и достигать баланса между подготовкой и игрой. Благодаря бесплатному доступу к официальным материалам и удобной навигации, этот ресурс становится незаменимым инструментом для тех, кто стремится к качественным и увлекательным игровым сессиям.

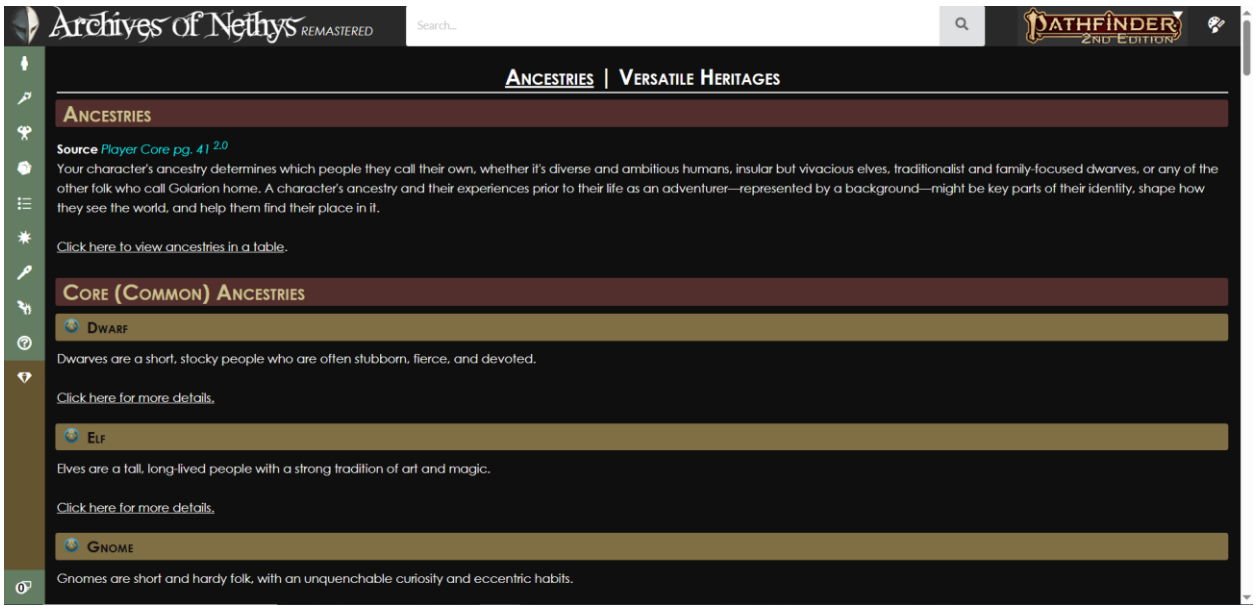

Рис. 3. PRD

Для сравнительного анализа конкурентов были выделены следующие критерии: доступность в России, бесплатность, наличие нескольких языков, удобство навигации, редактирование страниц, возможность обсуждения, поддержка мультимедиа, история изменений, рейтинг, рекомендации и обновление контента. Сравнительный анализ приложений аналогов по выделенным критериям представлен в таблице 1.

Таблица 1

Аналоги БЗ

Критерий	Rol20	Pathfinder	PRD
Доступность в России	+	+	+
Наличие нескольких языков	-	+- (имеет статьи на других языках)	-
Удобство навигации	+	+-	+
Возможность редактирования страниц	-	+	-

Возможность обсуждения	-	+	-
Поддержка мультимедиа	+ (изображения, интеграция с игровым столом)	+ (изображения, карты, видео)	+- (изображения, но меньше интерактивности)
История изменений	-	+	-
Рейтинг и рекомендации	-	+	-
Обновление контента	+ (официальный контент)	+- (официальный + пользовательский контент)	+ (официальный контент)
Бесплатный функционал	+- (бесплатный доступ к базовым правилам, но для полного контента требуется покупка)	+	+

На основе сравнительной таблицы можно выделить **ключевые критерии**, которые наиболее важны при проектировании и оценке базы знаний по настольным ролевым играм:

1. **Доступность в России** – Обеспечивает возможность использования ресурса без ограничений со стороны локального законодательства или провайдеров.
2. **Поддержка нескольких языков** – Позволяет охватить более широкую аудиторию, включая пользователей, не владеющих английским языком.
3. **Удобство навигации** – Критически важно для быстрого доступа к нужной информации, особенно в процессе подготовки к игре или во время сессии.

4. **Поддержка мультимедиа** – Возможность интеграции изображений, карт, видео и других визуальных материалов делает восприятие информации более наглядным и удобным.

5. СТЕЙКХОЛДЕРЫ ПРОЕКТА

Для эффективной работы базы знаний по настольным ролевым играм необходимо чётко определить роли пользователей и участников проекта, их функции, уровень вовлечённости и зоны ответственности. Это позволяет наладить взаимодействие между техническими и контентными участниками платформы, а также обеспечить устойчивое развитие и поддержку системы. Ниже представлена таблица с ключевыми ролями, их функциями и ответственностями, а также описание того, какую пользу каждая группа стейкхолдеров получает от участия в проекте. Для этого составим таблицу 2 пользователей, которые будут работать с БЗ.

Таблица 2

Пользователи в

Роль	Функции	Степень участия	Ответственность за проект
Новые пользователи «новобранцы»	-Чтение статей и правил. -Задание вопросов в комментариях. -Изучение основ игр.	Низкая (пассивное потребление контента).	-Соблюдение правил сообщества. -Корректное поведение в обсуждениях.
Опытные пользователи «старички»	-Редактирование статей. -Ответы на вопросы новичков. -Создание нового контента.	Высокая (активное участие в развитии вики).	-Поддержание качества контента. -Помощь новичкам. -Соблюдение правил сообщества.

Гости сайта, которые ищут информацию	-Просмотр статей и правил. -Поиск информации по играм.	Очень низкая (одноразовое или редкое использование).	-Соблюдение правил сайта (если оставляют комментарии или регистрируются)
Разработчики	-Создание и поддержка функционала сайта. -Исправление ошибок. -Добавление новых функций.	Максимальная (техническая поддержка и развитие платформы).	-Обеспечение стабильной работы сайта. -Реализация новых функций. -Защита данных пользователей.
Администраторы-модераторы	-Модерация контента (редактирование страниц, комментариев). -Управление пользователями (назначение ролей, блокировка нарушителей). -Контроль за соблюдением правил сообщества.	Высокая (постоянное участие в управлении контентом и сообществом).	-Поддержание порядка на сайте. -Обеспечение качества контента. -Решение конфликтов между пользователями.
Дизайнеры	-Разработка дизайн-макетов и шаблонов страниц. -Создание удобного и	Разовое участие (на этапе создания сайта).	-Создание удобного и привлекательного дизайна. -Обеспечение

	эстетичного интерфейса.		соответствия дизайна целям проекта.

В результате анализа было определено, что стейкхолдеры получат от данной системы:

Разработчики – получат опыт в создании веб-приложений, а также навыки работы с современными технологиями, такими как NodeJS + express и БД PostgreSQL, и инструментами WebStorm, Git, Docker. Это поможет им развить профессиональные компетенции и создать функциональную платформу.

Дизайнеры – приобретут опыт создания дизайн-макетов сайта, включая разработку композиции, прототипирование и создание удобного интерфейса. Они научатся работать с такими программами, как Figma, Adobe Photoshop и Illustrator, что улучшит их навыки в области UX/UI.

Администраторы-модераторы – смогут модерировать контент, управлять пользователями и контролировать соблюдение правил сообщества. Это позволит им поддерживать порядок на платформе и обеспечивать качество контента.

Опытные пользователи («старички») – смогут редактировать статьи, создавать новый контент и помогать новичкам. Их активное участие будет способствовать развитию вики и укреплению сообщества.

Новые пользователи («новобранцы») – получат доступ к обширной базе знаний по настольным ролевым играм, включая правила, сценарии и описания миров. Они смогут задавать вопросы и получать поддержку от опытных участников.

Гости сайта – смогут быстро находить информацию по играм без необходимости регистрации. Это удобно для тех, кто хочет ознакомиться с правилами или найти конкретные материалы.

6. ТРЕБОВАНИЯ К ПРОЕКТУ

6.1. Функциональные требования

При разработке программного обеспечения функциональные требования определяют функции, которые должно выполнять все приложение или только один из его компонентов. Функциональные требования важны, поскольку они показывают разработчикам программного обеспечения, как должна вести себя система. Если система не соответствует функциональным требованиям, значит она не работает должным образом.

Выделенные функциональные требования:

1. Гость должен иметь возможность авторизоваться на сайте;
2. Гость должен иметь возможность зарегистрироваться на сайте.
3. Авторизованный пользователь должен иметь возможность просматривать страницы, оставлять комментарии, изменять содержание страниц, добавлять новые страницы, оставлять комментарии к конкретным страницам.
4. Администратор имеет все возможности авторизованного пользователя + модерация статей, пользователей, комментариев, оценок.

6.2. Нефункциональные требования

Нефункциональные требования

Нефункциональные требования определяют стандарты производительности и атрибуты качества программного обеспечения, например, удобство использования системы, эффективность, безопасность, масштабируемость и т.д. Нефункциональные требования важны, поскольку

они помогают разработчикам программного обеспечения определять возможности и ограничения системы, которые необходимы для разработки высококачественного программного обеспечения.

Выделенные нефункциональные требования:

1. Безопасность: Система должна быть защищена от несанкционированного доступа

2. Производительность: Система должна быть высокопроизводительной и способной обслуживать достаточное количество пользователей без ухудшения ее производительности.

3. Масштабируемость: Система должна иметь возможность увеличивать или уменьшать масштаб по мере необходимости.

4. Наличие: Система должна быть доступна, когда это необходимо.

5. Техническое обслуживание: Система должна быть проста в обслуживании и обновлении.

6. Переносимость: Система должна работать на разных платформах с минимальными изменениями.

7. Надежность: Система должна быть надежной и соответствовать требованиям пользователя.

8. Юзабилити: Система должна быть простой в использовании и понятной.

9. Совместимость: Система должна быть совместима с другими системами.

10.Юридические вопросы: Система должна соответствовать всем применимым законам и правилам.

11. Конфиденциальность: Сервис должен соответствовать федеральному закону о персональных данных от 27.07.2006 N 152-ФЗ.

7. ТЕХНОЛОГИИ

7.1. Базы данных (БД)

7.1.1. Выбор типа СУБД

Прежде чем приступить к проектированию структуры базы данных для вики-подобной системы, необходимо определить, какой тип СУБД лучше подходит для наших задач: **реляционная (SQL)** или **документо-ориентированная (NoSQL)**.

Ключевые критерии выбора:

1. **Гибкость данных** – насколько часто будет меняться структура (например, добавление новых полей в статьи).
2. **Масштабируемость** – ожидается ли большой рост данных и нагрузки.
3. **Сложность запросов** – нужны ли аналитические отчёты с множественными связями.
4. **Скорость разработки** – насколько важно быстро вносить изменения без миграций схемы.

На основе этих критериев проведём сравнительный анализ и определим, какая модель данных (SQL или NoSQL) лучше соответствует требованиям проекта.

Таблица 3

Сравнительный анализ SQL с NoSQL

№	Критерий	SQL (PostgreSQL, MySQL, MSSQL, …)	NoSQL (MongoDB, Radis, …)	Что подходит для вики?
1	**Структура данных**	Жёсткая схема,	Гибкая схема, документы в JSON-формате	**NoSQL** — статьи и комментарии имеют разную структуру,

		таблицы с отношениями		возможны вложенные данные
2	**Масштабируемость**	Вертикальное (+репликация)	Горизонтальное (шардинг)	**NoSQL** — проще масштабировать при росте трафика
3	**Скорость записи**	Зависит от индексов, транзакции	Быстрая вставка, оптимизирована под запись	**NoSQL** — частая публикация статей и правок
4	**Сложные запросы**	JOIN, вложенные подзапросы	Агрегации, ссылки между коллекциями	**SQL** — если нужны аналитические отчёты, **NoSQL** — если запросы простые
5	**Гибкость**	Изменение схемы требует миграций	Динамическое изменение структуры	**NoSQL** — удобно для MVP и частых изменений
6	**Транзакции**	Поддержка ACID (атомарность)	Мультидокументные транзакции (с v4.0)	**SQL** — если критична целостность, **NoSQL** — если редкие конфликты

При выборе типа базы данных для вики-подобной системы было рассмотрено два принципиально разных подхода: реляционные (SQL) и нереляционные (NoSQL) базы данных. После тщательного анализа требований проекта стало очевидно, что NoSQL-решение предпочтительнее по следующим ключевым причинам:

1. Гибкость структуры данных

NoSQL базы данных предлагают схему по требованию (schemaless), что критически важно для динамически развивающейся вики-системы:

- **Быстрая адаптация**: Добавление новых полей (например, тегов, метаданных, медиавложений) не требует сложных миграций схемы.

- **Разнородные данные**: Статьи, обсуждения и комментарии могут иметь совершенно разную структуру без необходимости создания множества связанных таблиц.
- **Вложенные данные**: Возможность хранить сложные иерархические объекты (например, комментарии внутри обсуждений) в одном документе.

2. Масштабируемость и производительность

NoSQL базы изначально разрабатывались для горизонтального масштабирования (шардинга), что делает их идеальными для высоконагруженных систем:

- **Распределённые данные**: Легко масштабируются на несколько серверов для обработки большого трафика.
- **Оптимизация под запись**: Быстрая вставка и обновление данных (например, частые правки статей).
- **Эффективное хранение**: Данные хранятся в формате, близком к объектной модели приложения (JSON/BSON), что уменьшает накладные расходы на преобразование.

3. Удобство для разработки

NoSQL значительно ускоряет процесс разработки и итераций:

- **Отсутствие JOIN**: Сложные связи заменяются вложенными документами или ссылками, что упрощает запросы.
- **Динамические запросы**: Возможность выбирать данные без жёсткой схемы (например, фильтрация по неиндексированным

7.1.2. Выбор СУБД

Для разработки базы знаний по настольным ролевым играм необходимо выбрать такую систему управления базами данных (СУБД), которая сможет эффективно обрабатывать полуструктурированные данные, обеспечивать масштабируемость и высокую скорость чтения, а также быть гибкой в изменении структуры данных. Учитывая специфику проекта — наличие большого количества текстового контента, категорий, комментариев и возможных вложенных структур — основными требованиями к СУБД являются:

- **Поддержка документоориентированной модели хранения**, так как статьи и связанные с ними данные имеют сложную вложенную структуру;
- **Гибкость схемы**, позволяющая вносить изменения без трудоёмких миграций;
- **Высокая производительность при чтении и масштабируемость**, особенно в условиях большого количества пользователей;
- **Удобство работы с вложенными данными**, такими как списки тегов, комментариев, изображений и прочего контента.

На основании этих критериев был проведён сравнительный анализ современных СУБД, представленный в таблице ниже.

Таблица 4

Сравнительный анализ СУБД

№	СУБД	Поддержка документоориентированной модели	Гибкость схемы	Масштабируемость и	Поддержка вложенн

				производительность	ых данных
1	**MongoDB**	Да	Высокая	Высокая (в т.ч. шардинг)	Да
2	**Cassandra**	Нет	Низкая	Очень высокая	Нет
3	**Redis**	Нет	Отсутствует	Очень высокая (in-memory)	Нет
4	**Firestore**	Частично	Средняя	Средняя (сильная в real-time)	Частично

На основе выдвинутых требований к системе и проведённого сравнительного анализа можно сделать вывод, что MongoDB полностью соответствует всем необходимым критериям. Её документо-ориентированная модель идеально подходит для хранения полуструктурированных данных, таких как статьи, категории и комментарии. Благодаря высокой гибкости схемы, поддержке вложенных структур и масштабируемости, MongoDB обеспечивает эффективное хранение, удобную организацию и быстрый доступ к информации без необходимости в сложных миграциях.

7.1.3. Коллекции БД

На основании предоставленной схемы был проведён анализ структуры базы данных для вики-системы, посвящённой настольным ролевым играм. Система предназначена для хранения и организации игрового контента, включая статьи, категории, обсуждения и историю изменений.

Основные сущности и их взаимосвязи

1. Категории (Category)

- **Поля**: id, title, description, parent_id, icon, created_at, created_by, updated_at
- **Назначение**: Иерархическая организация контента
- **Связи**:
  - Содержит статьи (Article)
  - Может иметь родительскую категорию (parent_id)

2. Статьи (Article)

- **Поля**: id, title, description, content, tags, views, discussion, icon, created_at, updated_at, created_by
- **Назначение**: Основной контент системы
- **Связи**:
  - Принадлежит категории (Category)
  - Имеет обсуждение (Discussion)
  - Содержит историю изменений (History)

3. Обсуждения (Discussion)

- **Поля**: id, title, short_description, description, comment
- **Назначение**: Обсуждение контента
- **Связи**:
  - Связано со статьёй (Article)
  - Содержит комментарии (Comment)

4. Комментарии (Comment)

- **Поля**: id_user, id_commit, id_block, text, rating, created_at, updated_at
- **Назначение**: Пользовательские обсуждения
- **Связи**:
  - Принадлежат обсуждению (Discussion)
  - Связаны с пользователем (User)

5. История изменений (History)

- **Поля**: id, article_id, category_id, user_id, version, changes, timestamp, status
- **Назначение**: Отслеживание изменений контента
- **Связи**:
    - Связана со статьёй (Article)
    - Связана с категорией (Category)
    - Связана с пользователем (User)

6. Каталоги (Catalog)

- **Поля**: id, title, category
- **Назначение**: Организация категорий
- **Связи**:
    - Содержит категории (Category)

7. Закладки (Bookmakers)

- **Поля**: user_id, article_id, saved_at
- **Назначение**: Персонализация пользовательского опыта
- **Связи**:
    - Связаны с пользователем (User)
    - Связаны со статьёй (Article)

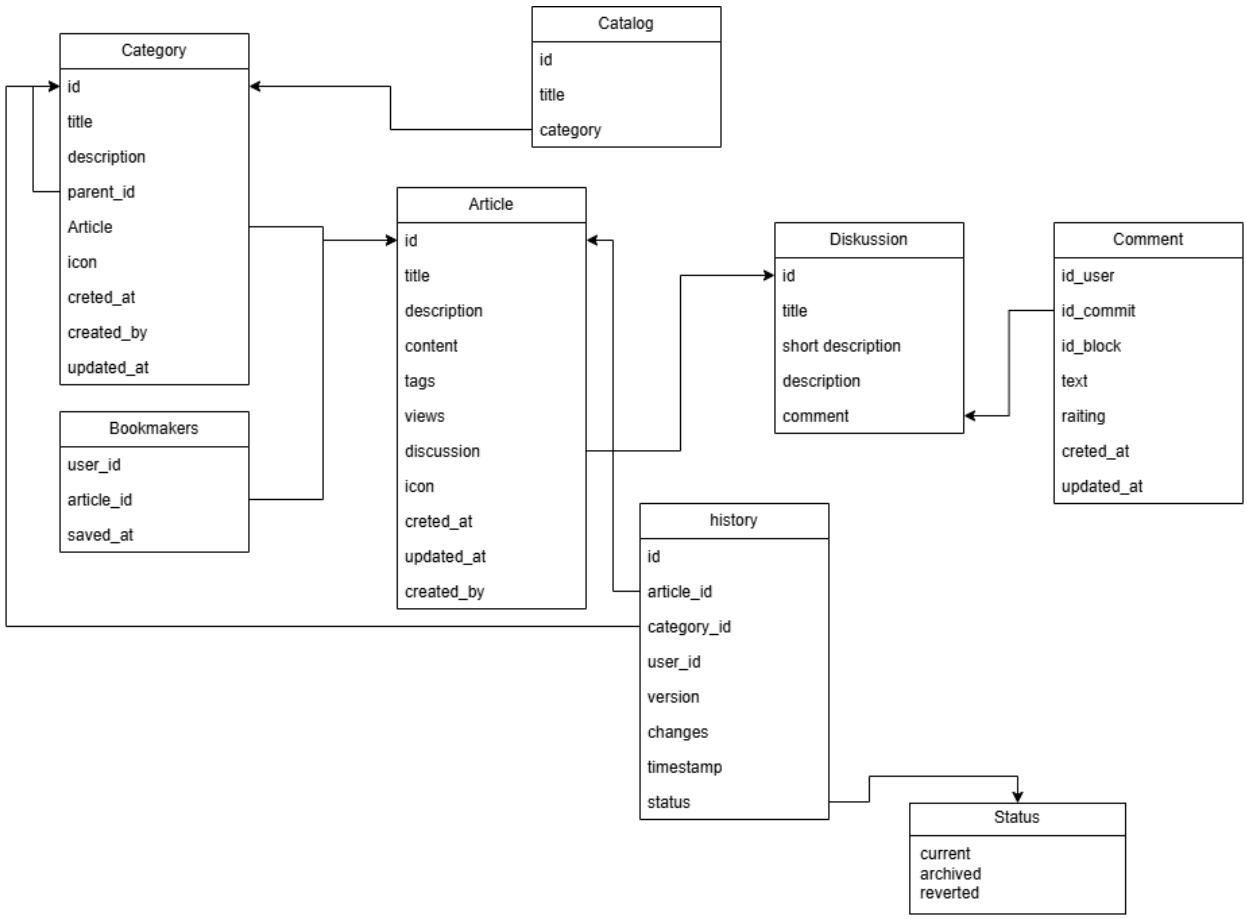

Рис. 4. Схема коллекций БД

7.2. Frontend (Клиентская часть)

React — библиотека для создания пользовательских интерфейсов, выбранная для реализации клиентской части приложения. Её компонентный подход позволяет разрабатывать модульные и переиспользуемые элементы интерфейса, что критически важно для динамических веб-приложений.

Преимущества выбора:

1. Виртуальный DOM — оптимизирует рендеринг, уменьшая нагрузку на браузер и повышая производительность при частых обновлениях интерфейса (например, изменение количества товаров в корзине).
2. Экосистема — интеграция с популярными библиотеками (Redux для управления состоянием, React Router для навигации, Material-UI для готовых компонентов) ускоряет разработку.
3. JSX — синтаксис, сочетающий HTML и JavaScript, упрощает создание интерактивных элементов.
4. Сообщество — обширная документация и регулярные обновления гарантируют долгосрочную актуальность.

Конкурентный анализ:

- Angular: Фреймворк с жёсткой структурой и встроенными решениями (например, двухстороннее связывание данных). Однако его сложность и низкая гибкость замедляют разработку для небольших команд.
- Vue.js: Прост в изучении, но уступает React в экосистеме и поддержке корпоративного уровня.

Итог: React обеспечивает оптимальный баланс между гибкостью, производительностью и экосистемой, что критично для интернет-магазина с динамическим контентом.

8. ОПИСАНИЕ ФУНКЦИОНАЛА ПРИЛОЖЕНИЯ "БАЗА ЗНАНИЙ" ДЛЯ СЕРВИСА НРИ "ROLE FORGE"

База знаний предоставляет удобный доступ к структурированным материалам: правилам, описаниям миров, персонажам, предметам и другим элементам игровых систем. Основной задачей интерфейса является обеспечение простой и интуитивной навигации по вложенным разделам — от каталогов до отдельных страниц — с акцентом на визуальную подачу, читаемость и быстроту доступа к нужной информации. Данный раздел содержит описание ключевых элементов пользовательского интерфейса и функциональных компонентов системы, иллюстрируя, как реализована структура базы знаний и взаимодействие с её содержимым. Рис. 5. отображает приветственную часть на главной странице базы знаний.

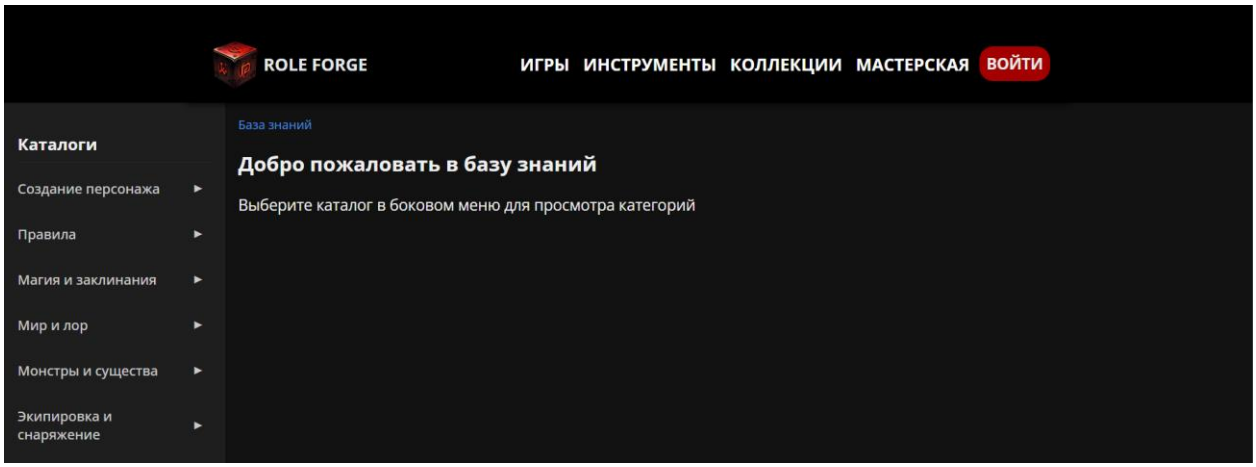

Рис. 5. Главная страница

Используя боковую панель навигации, пользователь может перейти к конкретным каталогам, содержащим визуальные карточки с ссылками на соответствующие категории. При выборе категории отображаются связанные с ней страницы, представленные в виде карточек с изображениями и заголовками. Навигация по базе знаний реализована с помощью визуальных элементов и хлебных крошек, что обеспечивает интуитивное перемещение по

иерархии — от каталога к категории и далее к отдельной странице с подробным описанием и иллюстрациями.

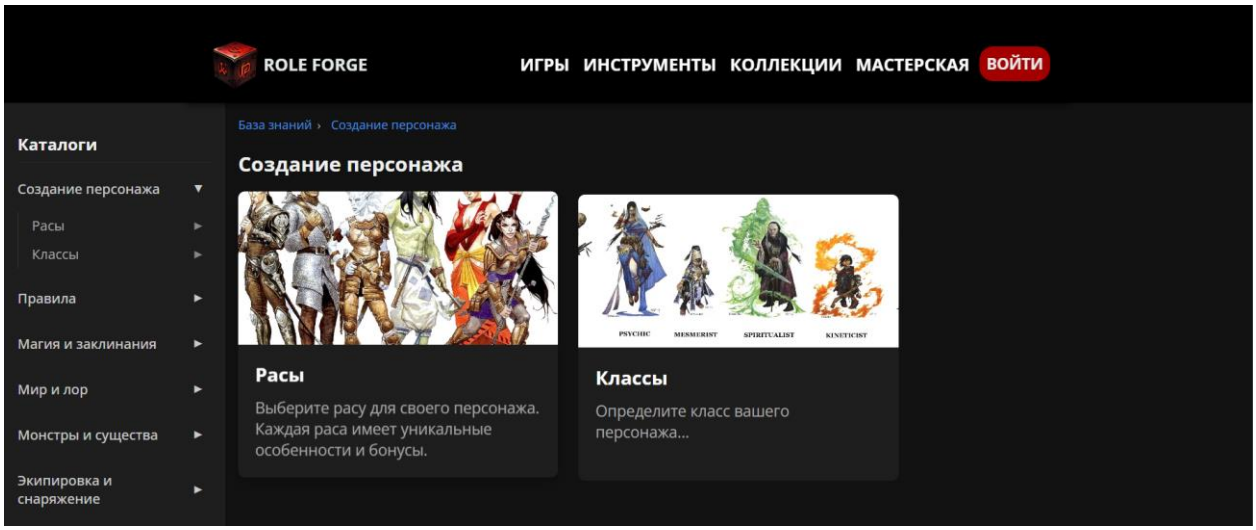

Рис. 6. Выбор каталога на странице

При выборе каталога в боковом меню в основной области страницы отображается его название и набор категорий, представленных в виде карточек с изображением, названием и описанием; каждая карточка кликабельна и позволяет перейти к списку страниц, относящихся к выбранной категории.

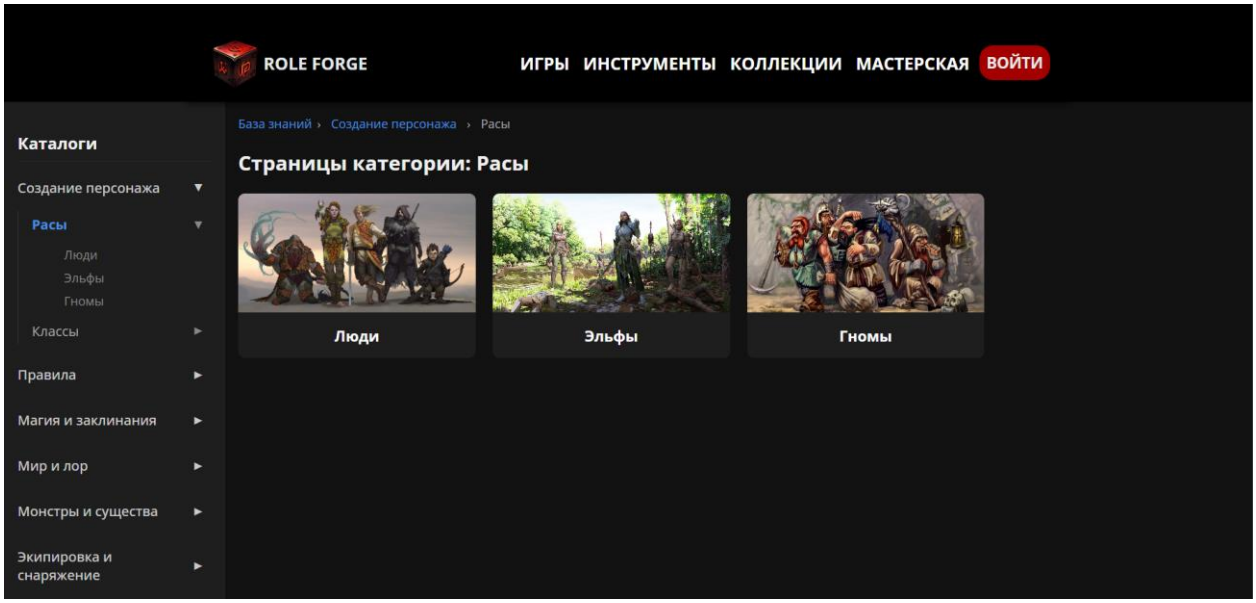

Рис. 7. Выбор категории на странице

При выборе конкретной категории на странице отображаются карточки страниц этой категории в виде сетки, каждая из которых содержит изображение, название и ведёт на отдельную страницу с подробной информацией; визуально карточки оформлены в виде плиток с картинкой сверху и заголовком снизу, создавая удобную и наглядную структуру для выбора нужного материала.

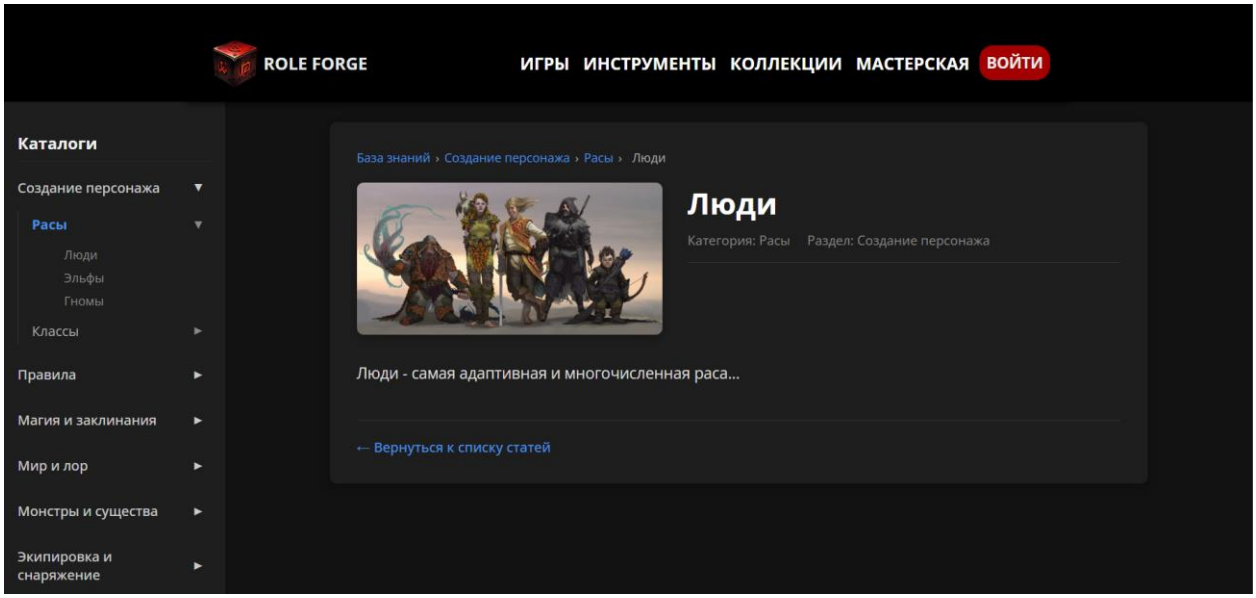

Рис. 8. Страница статей

При переходе на отдельную страницу статьи пользователь видит полное содержимое выбранной статьи из базы знаний, включая заголовок, изображение, категорию и раздел, к которому она относится; навигация остаётся доступной через боковое меню и хлебные крошки, позволяющие вернуться к каталогу или категории, а если страница открыта напрямую, система автоматически находит её в структуре данных и отображает соответствующую информацию.

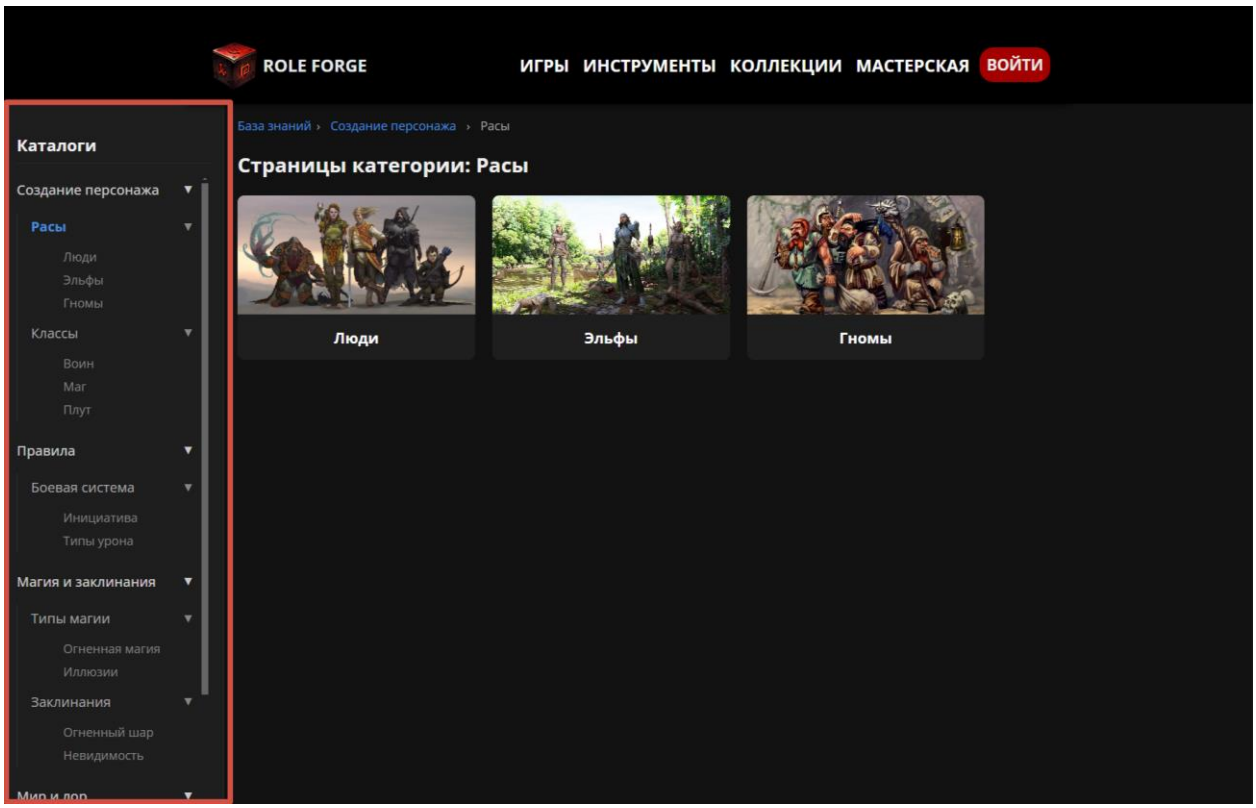

Рис. 9. Компонент боковая панель

Компонент бокового меню базы знаний отображает список каталогов, которые можно разворачивать и сворачивать, показывая вложенные категории, а в категориях — страницы; при клике на каталог, категорию или страницу компонент обновляет своё состояние и вызывает функцию onCatalogSelect (если она передана) с текущим выбранным каталогом, категорией и страницей, что позволяет синхронизировать выбор с остальной частью приложения; кроме того, если onCatalogSelect отсутствует, навигация происходит через navigate с передачей состояния, чтобы обеспечить правильный переход и отображение выбранного раздела; при этом активные элементы визуально выделяются, а открытие и закрытие разделов управляется локальным состоянием с помощью флагов расширения.

9. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

9.1. Главная страница с отображением каталогов и категорий

Компонент **knowledgeBase.jsx** (БазаЗнаний) отвечает за то, какой каталог и категорию сейчас просматривает пользователь в базе знаний, и что при этом должно отображаться на экране.

```
useEffect(() => {
  if (location.state?.selectedCatalog) {
    setSelectedCatalog({
      ...location.state.selectedCatalog,
      selectedCategory: location.state.selectedCategory || null
    });
  } else {
    setSelectedCatalog(null);
  }
}, [location.state]);
```

Внутрь данного компонента встроен компонент с навигационным меню, код встраивания представлен ниже:

```
<KnowledgeBaseSidebar
  onCatalogSelect={setSelectedCatalog}
  selectedCatalog={selectedCatalog}
/>
```

В основной части страницы через хлебные крошки (пути от главной страницы до страницы текущего уровня) и условные блоки выводится либо приветствие (если ничего не выбрано), либо список категорий выбранного каталога:

```
{!selectedCatalog.selectedCategory ? (
  <div className="kb-categories-grid">
    {selectedCatalog.categories.map(category => (
      <div
        key={category.id}
        className="kb-category-card"
        onClick={() => setSelectedCatalog({...selectedCatalog, selectedCategory: category})}
      >
        ...
      </div>
    ))}
  </div>
) : (
  ...
)}
```

Вот как это работает простыми словами:

1. **Когда пользователь переходит на страницу**, компонент проверяет, был ли уже выбран какой-то каталог и категория (через location.state). Если да — сохраняет их, чтобы знать, что показать.
2. **Слева отображается боковая панель** с каталогами (KnowledgeBaseSidebar). Когда пользователь кликает по каталогу или категории, компонент обновляет текущее состояние — то есть запоминает, какой каталог и категорию выбрал пользователь.
3. **Если выбран только каталог, но ещё не категория**, в основной части экрана показываются **карточки категорий** внутри этого каталога. Пользователь может кликнуть по любой карточке, чтобы перейти к нужной категории.
4. **Если выбрана и категория**, тогда вместо категорий показываются **карточки отдельных страниц** (статей), которые входят в эту категорию. Каждая карточка ведёт на отдельную страницу с полной информацией.

Таким образом, компонент синхронизирует выбор каталога и категории через состояние и URL, управляет отображением и навигацией по базе знаний.

9.2. Страница с отображением конкретных статей базы знаний

Этот компонент отвечает за **отображение одной конкретной статьи** из базы знаний. Он делает так, чтобы страница корректно открывалась как при переходе через интерфейс, так и при прямом вводе URL в адресной строке.

Этот компонент отображает отдельную статью базы знаний, извлекая из URL параметр id статьи и пытаясь инициализировать состояние из location.state (если пользователь пришёл с навигацией, передающей каталог и категорию):

```
useEffect(() => {
  const catalogFromState = location.state?.selectedCatalog;
  const categoryFromState = location.state?.selectedCategory;

  if (catalogFromState && categoryFromState) {
    const foundPage = categoryFromState.pages.find(p => p.id === pageId);
    setSelectedCatalog({
      ...catalogFromState,
      selectedCategory: categoryFromState,
    });
    setPage(foundPage || null);
  }
}, [location.state, pageId]);
```

Если данные из location.state отсутствуют (например, при обновлении страницы), происходит обход всего JSON с каталогами и категориями для поиска нужной страницы по id:

```
useEffect(() => {
  if (!page && data.catalogs) {
    for (const catalog of data.catalogs) {
      for (const category of catalog.categories) {
        const foundPage = category.pages.find(p => p.id === pageId);
        if (foundPage) {
          setSelectedCatalog({
            ...catalog,
            selectedCategory: category,
          });
          setPage(foundPage);
          return;
        }
      }
    }
  }
}, [page, pageId]);
```

Если страница не найдена, выводится сообщение об ошибке с ссылкой возврата. Если страница есть, показывается боковая панель навигации с текущим выбором каталога и категории, а в основной части — хлебные

крошки для навигации вверх по иерархии и контент статьи с изображением, названием, метаданными и текстом:

```
<article className="kb-article-content">
  <div className="kb-breadcrumbs">
    <Link to="/knowledgebase">База знаний</Link>
    <span> › </span>
    <Link to="/knowledgebase" state={{ selectedCatalog, selectedCategory: null }}>
      {selectedCatalog.title}
    </Link>
    <span> › </span>
    <Link to="/knowledgebase" state={{ selectedCatalog, selectedCategory: selectedCatalog.selectedCategory }}>
      {selectedCatalog.selectedCategory.name}
    </Link>
    <span> › </span>
    <span>{page.title}</span>
  </div>
  ...
</article>
```

Вот как он работает простыми словами:

1. **Когда пользователь переходит на страницу статьи через интерфейс**, вместе с ним передаётся информация о выбранном каталоге и категории (location.state). Компонент использует это, чтобы:
    - Запомнить текущий каталог и категорию.
    - Найти нужную статью по её id, которая берётся из адреса страницы (pageId).
2. **Если пользователь просто обновил страницу или перешёл по прямой ссылке**, location.state уже нет. Тогда компонент **сам ищет статью внутри всей структуры данных**, перебирая все каталоги и категории:
    - Как только находит нужную статью, сохраняет в состоянии текущий каталог, категорию и саму статью.
3. **Если нужная статья не найдена**, компонент показывает сообщение об ошибке и предлагает вернуться назад.
4. Если всё найдено, то:

- **Слева отображается боковая панель** с каталогами.
- **В основной части экрана** — хлебные крошки, позволяющие подняться назад к каталогу или категории.
- Ниже — **контент статьи**: заголовок, изображение, текст и другая информация.

5. Также есть кнопка "назад", чтобы быстро вернуться к списку всех статей выбранной категории.

Таким образом, компонент обеспечивает корректное отображение статьи с учётом переходов и перезагрузок страницы.

9.3. knowledgeBaseArticle

Этот компонент — боковое меню (sidebar) базы знаний, которое отображает иерархию каталогов, категорий и страниц, позволяя разворачивать и сворачивать разделы, а также выбирать элементы для просмотра.

9.3.1. Инициализация раскрытых элементов при смене выбора

Если пользователь выбрал каталог или категорию (например, перешёл по ссылке), то нужные разделы в меню автоматически разворачиваются:

```
useEffect(() => {
  if (selectedCatalog?.id) {
    setExpandedCatalogs(prev => ({
      ...prev,
      [selectedCatalog.id]: true
    }));
  }

  if (selectedCatalog?.selectedCategory?.id) {
    setExpandedCategories(prev => ({
      ...prev,
      [selectedCatalog.selectedCategory.id]: true
    }));
  }
}, [selectedCatalog]);
```

9.3.2. Переключение раскрытия каталога

При клике по каталогу:

- Каталог разворачивается или сворачивается.
- Сбрасывается выбор категории и страницы.
- Сохраняется текущий выбор:

```
const toggleCatalog = (catalogId) => {
  setExpandedCatalogs(prev => ({
    ...prev,
    [catalogId]: !prev[catalogId]
  }));

  const catalog = data.catalogs.find(c => c.id === catalogId);
  if (catalog && onCatalogSelect) {
    onCatalogSelect({
      ...catalog,
      selectedCategory: null,
      selectedPage: null,
    });
  } else {
    navigate('/knowledgebase');
  }
};
```

9.3.3. Переключение раскрытия категории

Аналогично для категории — раскрытие переключается, вызывается `onCatalogSelect` с выбранной категорией и без выбранной страницы:

```
const toggleCategory = (catalog, category) => {
  setExpandedCategories(prev => ({
    ...prev,
    [category.id]: !prev[category.id]
  }));

  if (onCatalogSelect) {
    onCatalogSelect({
      ...catalog,
      selectedCategory: category,
      selectedPage: null,
    });
  } else {
    navigate('/knowledgebase', {
      state: {
        selectedCatalog: catalog,
        selectedCategory: category,
        selectedPage: null,
      }
    });
```

```
  }
};
```

9.3.4. Выбор страницы

При клике по странице она сохраняется и отображается на экран.

```
const handlePageClick = (catalog, category, page) => {
  if (onCatalogSelect) {
    onCatalogSelect({
      ...catalog,
      selectedCategory: category,
      selectedPage: page,
    });
  } else {
    navigate('/knowledgebase', {
      state: {
        selectedCatalog: catalog,
        selectedCategory: category,
        selectedPage: page,
      }
    });
  }
};
```

9.3.5. Подсветка активных элементов

Если текущая категория или страница выбрана, она получает CSS-класс active, чтобы визуально выделяться:

```
const isCategoryActive = (category) => {
  return selectedCatalog?.selectedCategory?.id === category.id;
};

const isPageActive = (page) => {
  return selectedCatalog?.selectedPage?.id === page.id;
};
```

9.3.6. Отрисовка

- Все каталоги берутся из JSON-данных.
- При нажатии по заголовку каталога — он раскрывается и показывает список категорий.
- При раскрытии категории — показываются страницы внутри.
- Все клики обновляют текущее состояние и позволяют быстро перемещаться по материалам.

Это обеспечивает удобную навигацию по структуре базы знаний с визуальной индикацией текущего выбора и возможностью быстро переключаться между разделами.

10. СПИСОК ЛИТЕРАТУРЫ

1. ReactJS Official Documentation [Электронный ресурс]: документация. – Режим доступа: https://reactjs.org/docs/getting-started (дата обращения 15.04.2025).

2. How to use React useState and useEffect hooks [Электронный ресурс]: статья. – Режим доступа: https://www.freecodecamp.org/news/react-usestate-hook/ (дата обращения 29.04.2025).

3. React Router: Declarative Routing for React.js [Электронный ресурс]: документация. – Режим доступа: https://reactrouter.com/en/main (дата обращения 21.04.2025).

4. Working with MongoDB in React [Электронный ресурс]: блог. – Режим доступа: https://www.mongodb.com/languages/react (дата обращения 10.05.2025).