

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

К защите допустить:

Заведующая кафедрой информ-
матики

_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

**ОНЛАЙН-ПЛАТФОРМА ДЛЯ ИЗУЧЕНИЯ ИНОСТРАННЫХ
ЯЗЫКОВ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИЙ REACT И ПЛАТ-
ФОРМЫ .NET**

БГУИР ДП 1-40 04 01 061 ПЗ

Студент

А. А. Круглая

Руководитель

В. С. Плиска

Консультанты:

*от кафедры информатики
по экономической части*

В. С. Плиска
Т. А. Рыковская

Нормоконтролер

Н.Н. Бабенко

Рецензент

Минск 2023

РЕФЕРАТ

ОНЛАЙН-ПЛАТФОРМА ДЛЯ ИЗУЧЕНИЯ ИНОСТРАННЫХ ЯЗЫКОВ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ REACT И ПЛАТФОРМЫ .NET: дипломный проект / А.А. Круглая. – Минск: БГУИР, 2023, – п.з. – 111 с., схем/плакатов– 2 л.

Ключевые слова: ОНЛАЙН-ПЛАТФОРМА, СЕРВЕРНОЕ API, GOOGLE TRANSLATE API, C#, REACT, VISUAL STUDIO, AZURE CLOUD, BACKEND, FRONTEND, ASP.NET.

Объектом разработки является онлайн-платформа для изучения иностранных языков с использованием технологий React и платформы .NET.

Целью проекта является проектирование и разработка React-приложения и серверного API для взаимодействия с базой данных. Его главной задачей является предоставление удобного и эффективного инструмента для расширения словарного запаса и развития навыков чтения на иностранном языке.

При разработке приложения использовался следующий набор технологий: REACT, ENTITY FRAMEWORK, GOOGLE TRANSLATE API, AZURE CLOUD STORAGE, C#, ASP.NET, FIREBASE AUTHENTICATION, FIREBASE ANALYTICS, JWT.

В первом разделе данной работы осуществляется анализ востребованности разрабатываемого приложения, анализ существующих аналогов и формирование требований к приложению.

Второй раздел содержит информацию о технологиях, используемых для разработки React-приложения и серверного API.

В третьем разделе приведён процесс проектирования архитектуры приложения, его бэкенда, проектирования базы данных.

Четвертый раздел содержит описание процесса разработки функционала приложения и бэкенда.

Пятый раздел содержит описание процесса тестирования основной функциональности приложения.

В шестом разделе приведено технико-экономическое обоснование эффективности разработки приложения.

В разделе выводы содержатся итоги проделанной работы по данному дипломному проекту.

Дипломный проект является завершённым, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития приложения и наращивания его функционала.

Министерство образования Республики Беларусь
Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ

Факультет КС и С Кафедра информатики
Специальность 1-40 04 01 Специализация 00

УТВЕРЖДАЮ
Н.А.Волорова

« » 20 г.

ЗАДАНИЕ

по дипломному проекту студента

Круглой Анастасии Алексеевны

(фамилия, имя, отчество)

1. Тема проекта: **Онлайн-платформа для изучения иностранных языков с применением технологий React и платформы .NET**

утверждена приказом по университету от « 17 » марта 2023 г. № 63-с

2. Срок сдачи студентом законченной работы 01 июня 2023 года

3. Исходные данные к проекту Тип операционной системы – ОС Windows;
языки программирования – JS, C#; база данных - PostgreSQL; интегрированные среды раз-
работки - Visual Studio; распределённая система управления версиями - Git;

Назначение разработки: автоматизация процессов изучения иностранных языков
посредством использования онлайн-платформы

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Введение

1 Анализ предметной области, аналогов и формирование требований к проекту

2 Обзор используемых в проекте технологий

3 Проектирование программного средства

4 Создание программного средства

5 Тестирование программного средства

6 Технико-экономическое обоснование

Заключение

Список использованных источников

Приложение А Текст программы

5. Перечень графического материала (с точным указанием наименования) и обозначения вида и типа материала) _____

Схема базы данных – Плакат – формат А4, лист 1.

Взаимодействие компонентов приложения – Плакат – формат А4, лист 1.

Диаграмма вариантов использования – Плакат – формат А4, лист 1.

6. Содержание задания по технико–экономическому обоснованию

1 Расчёт затрат на разработку онлайн-платформы для изучения иностранных языков

2 Оценка экономической эффективности проекта

Задание выдал _____ / Т.А. Рыковская /

Задание выдал: _____

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта (работы)	Объём этапа в %	Срок выпол- нения этапа	Примеча- ние
Анализ предметной			
области, разработка технического задания	15-20	01.02–14.02	
Разработка функциональных требований, проектирование архитектуры программы	20-15	15.02–06.03	
Разработка схемы программы, алгоритмов, схемы данных	20-15	07.03–27.03	
Разработка программного средства	15-20	28.03–24.04	
Тестирование и отладка	10	25.04–08.05	
Оформление пояснительной записки и графического материала	20	09.05–31.05	

Дата выдачи задания 1 февраля 2023 г. Руководитель _____ /В.С. Плискин /

Задание принял к исполнению _____ / А.А. Круглая /

СОДЕРЖАНИЕ

Определения и сокращения.....	7
ВВЕДЕНИЕ.....	9
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, АНАЛОГОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТУ	10
1.1 Анализ предметной области.....	10
1.2 Анализ веб-приложений для изучения иностранных языков	12
1.3 Эффективность выбранной системы запоминания.....	16
1.4 Требования к проектируемому программному средству	16
2 ОБЗОР ИСПОЛЬЗУЕМЫХ В ПРОЕКТЕ ТЕХНОЛОГИЙ.....	18
2.1 Выбор платформы приложения	18
2.2 Интегрированная среда разработки.....	20
2.3 Язык программирования C# и платформа .NET	21
2.4 ASP.NET	23
2.5 PostgreSQL.....	24
2.6 Entity Framework (EF) Core.....	25
2.7 JavaScript.....	26
2.8 React	27
2.9 Google Translate API	28
2.10 Azure Cloud и Azure Blob.....	29
2.11 Firebase.....	30
2.11.1 Firebase Authentication.....	30
2.11.2 Firebase Analytics	31
3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА.....	32
3.1 Формирование функциональных требований	32
3.2 Model-View-Controller (MVC) шаблон	33
3.3 React-приложение	34
3.4 Серверное API.....	45
3.5 База данных	52
4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА.....	54
4.1 Элементы пользовательского интерфейса	54
4.2 Регистрация и авторизация пользователя	57
4.3 Работа с текстовыми документами.....	59

4.4	Работа с флеш-карточками	61
5	ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА.....	65
5.1	Тестирование корректности создания карточки	65
5.2	Тестирование создания пользователя.....	66
5.3	Добавление книги	68
6	ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ОНЛАЙН-ПЛАТФОРМЫ ДЛЯ ИЗУЧЕНИЯ ИНОСТРАННЫХ ЯЗЫКОВ	70
6.1	Характеристика онлайн-платформы, разрабатываемой для реализации на рынке	70
6.2	Расчет инвестиций в разработку онлайн-платформы для её реализации на рынке	71
6.3	Расчет экономического эффекта от реализации онлайн-платформы для изучения иностранных языков на рынке.....	74
6.4	Расчет показателей экономической эффективности разработки и реализации онлайн-платформы для изучения иностранных языков на рынке	75
	ЗАКЛЮЧЕНИЕ	78
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	78
	ПРИЛОЖЕНИЕ А.....	80

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В данной пояснительной записке применяются следующие определения и сокращения:

Онлайн-платформа – веб-сайт или приложение, которое предоставляет пользователю определенный сервис, функционал или возможность взаимодействия в интернете. Она служит центральным местом, где пользователи могут получать доступ к различным ресурсам, услугам или информации через интернет.

Архитектура приложения – принципиальная организация приложения, воплощенная в его элементах, их взаимоотношениях друг с другом и со средой, а также принципы, направляющие его проектирование и эволюцию.

Паттерн – повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Аутентификация – проверка подлинности предъявленного пользователем идентификатора.

Авторизация – предоставление определённого лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Фреймворк – программное обеспечение, предназначенное для облегчения процессов разработки и поддержки компонентов программного проекта.

Фронтенд – термин, используемый в веб-разработке для обозначения части веб-приложения или веб-сайта, которая отвечает за визуальное представление и взаимодействие с пользователем. Он относится к клиентской стороне разработки, которая работает непосредственно в браузере пользователя.

Бэкенд – серверная часть исполняемого кода системы в архитектуре клиент-сервер.

СУБД – система управления базами данных.

API – Application Programming Interface – прикладной программный интерфейс.

HTTP – HyperText Transfer Protocol – протокол передачи гипертекста.

EF – Entity Framework.

ORM – Object–Relational Mapping – объектно-реляционное отображение.

SCSS (Sassy CSS) – расширение языка CSS, которое добавляет дополнительные возможности и функциональность к стандартному CSS. SCSS является препроцессором CSS, что означает, что он позволяет использовать дополнительные конструкции и синтаксис, которые не поддерживаются непосредственно в CSS.

Браузер – программное обеспечение, которое позволяет пользователям просматривать и взаимодействовать с веб-страницами и другими ресурсами в Интернете.

Система Лейтнера – методика эффективного запоминания и повторения информации, основанная на принципах активного повторения и распределения повторений во времени. Она была разработана немецким психологом Себастьяном Лейтнером в середине XX века и широко применяется в обучении и изучении различных предметов и языков.

Флеш-карточки – учебные карточки, которые используются для запоминания информации и повторения материала. Они представляют собой небольшие карточки с одной стороны которых написано определение, вопрос или задача, а с другой стороны - соответствующий ответ или решение.

MVC – архитектурный шаблон, который воплощает в себе три части: модель, представление и контроллер.

ВВЕДЕНИЕ

В современном мире все больше людей стремятся освоить иностранные языки, будь то в рамках профессионального развития, учебных целей или для расширения коммуникационных возможностей. Знание иностранного языка становится все более ценным в мировом рынке труда. Будь то в сфере бизнеса, международных отношений, туризма или научных исследований, умение общаться на нескольких языках дает преимущество и открывает двери к новым карьерным возможностям. Изучение иностранного языка развивает уверенность в себе, способность к адаптации и умение налаживать контакты с людьми из разных стран и культур. Это помогает расширить круг общения и улучшить межличностные отношения.

Однако, изучение нового языка может представлять определенные трудности, особенно когда речь идет о чтении книг на иностранном языке, где встречаются незнакомые слова и выражения.

Целью данного дипломного проекта является разработка веб-приложения, которое облегчит процесс чтения книг на иностранном языке и поможет пользователям улучшить свой словарный запас. Главной особенностью приложения является возможность создания флеш-карточек с переводом неизвестных иностранных слов, что позволит пользователям эффективно запоминать новые лексические единицы.

Приложение будет предоставлять доступ к широкой библиотеке книг на различных иностранных языках, предоставлять удобный интерфейс для чтения и одновременного создания флеш-карточек.

Разработка данного веб-приложения позволит пользователям эффективно изучать новые слова и выражения, повышая их языковую компетенцию и уверенность в чтении на иностранном языке. Кроме того, приложение будет доступно с любого устройства с подключением к интернету, что делает его удобным инструментом для самообразования в любом месте и в любое время.

Ожидается, что результатом данного дипломного проекта будет полнофункциональное веб-приложение, способное облегчить процесс чтения книг на иностранном языке и значительно улучшить словарный запас пользователей. Такое приложение будет иметь потенциал для применения в образовательных учреждениях, языковых курсах и среди самоучек, помогая им достичь большего успеха в изучении иностранных языков.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, АНАЛОГОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТУ

Конечный результат разработки и реализации программного средства закладывается на этапе подготовки, во время которого важно, как можно четче и детализировано определить особенности проекта.

Первый шаг на этапе подготовки, который необходимо выполнить, – формулирование проблемы, которую должно решить программное средство. Основная цель, которую преследует этап подготовки, – уменьшение уровня риска, так как адекватное планирование позволяет обнаружить и исключить возможные аспекты риска до начала этапа конструирования, что в свою очередь дает возможность выполнить большую часть проекта с минимальными потерями эффективности.

Считается, что неудачная выработка требований – это главный фактор риска в создании ПО, так как они подробно описывают, что должна делать программная система.

Перед тем как сформулировать необходимые требования следует изучить ряд вопросов, которые непосредственно окажут влияние не все дальнейшие этапы разработки.

В частности, необходимо провести анализ предметной области, для которой разрабатывается онлайн-платформа. По результатам анализа можно будет составить техническое задание к проектируемому программному средству, опираясь на которое будут сформулированы функциональные требования.

1.1 Анализ предметной области

Изучение иностранного языка в современном мире – это один из важных составляющих элементов в жизни современного, образованного человека. Знание иностранного языка не просто желательно, оно необходимо. Сегодня есть немало людей, желающих знать иностранный язык, потому что это знание дает им новые возможности и делает богаче их духовный мир.

В идеале успешный, амбициозный человек должен владеть несколькими иностранными языками и постоянно их улучшать, потому что изучение иностранных языков – совсем не быстрый процесс. Люди занимаются этим творческим и увлекательным занятием, чтобы:

- развивать мировоззрение;
- совершенствовать логическое мышление;
- повышать самооценку и уверенность в себе;
- иметь возможность найти понимание в любой точке мира;
- уметь выражать свои мысли кратко и четко.

В современном мире, где времени постоянно не хватает, нужно построить процесс получения новых знаний так, чтобы он помогал охватывать сразу

несколько целей в жизни. Изучение иностранных языков посредством создания карточек для запоминания в процессе чтения – идеальный подход для людей, которые хотят не только расширять свой словарный запас, но и больше читать. Ведь чтение как ни что другое помогает человеку получать новые опыт и знания. Чем активнее вы читаете, тем лучше понимаете носителей языка. Благодаря чтению можно выучить различные английские выражения и фразы, также появляется лучшее понимание структуры предложения.

Из основных преимуществ выбора чтения как способа изучения иностранных языков можно выделить следующие:

- чтение увеличивает словарный запас;
- чтение улучшает навык письма;
- чтение улучшает языковое чутье;
- чтение увеличивает багаж знаний;

Таким образом можно сделать вывод, что чтение книг на иностранном языке можно считать одним из самых эффективных и удобных способов их изучения, особенно если в дополнение к этому имеется возможность быстро и без усилий переводить неизвестные слова.

1.1.1 Анализ рынка веб-приложений для образования

На данный момент веб-приложения стали неотъемлемой частью нашей жизни. Сегодня эта разновидность онлайн-инструментов используется для различных видов задач:

- оптимизация бизнес-процессов;
- продажа товаров и услуг;
- распространение информации;
- общение пользователей друг с другом.

Все это достигается благодаря тому, что веб-приложения имеют ряд преимуществ, таких как доступность, эффективная разработка, простота для пользователя и масштабируемость. Доступность позволяет иметь доступ к веб-приложениям из всех браузеров, с различных типов устройств и в любое время. Процесс разработки веб-приложения относительно прост и экономически эффективен. Как известно веб-приложения не требуют от пользователей загрузки, что делает их легко доступными и избавляет пользователей от необходимости в обслуживании и ограничении емкости жесткого диска. Также они всегда актуальны и меньше подвержены риску нарушений безопасности за счет автоматического обновления программного обеспечения и безопасности. В дополнение ко всему компании, использующие веб-приложения, могут добавлять пользователей по мере необходимости, без дополнительной инфраструктуры и дорогостоящего оборудования.

Образование является одной из крупнейших отраслей в мире, на которую приходится более 6% мирового ВВП. Объем мирового рынка образования в 2021 году, по данным из открытых источников, достиг отметки 6,5 трлн долл. США.

В тоже время к наиболее стимулирующим рынок онлайн-образования факторам относятся гибкость и удобство обучения, а также актуальность образовательных программ, что было подтверждено данными исследования компании «Learning House» в 2019 году, по результатам которого 63% опрошенных выбрали онлайн-программы в связи с тем, что такой формат обучения проще всего совмещать с работой и личной жизнью, 34% сообщили, что в целом предпочитают такой метод обучения, а 3% заявили, что необходимая им программа обучения доступна только в онлайн-формате.

Прогнозируется, что к 2025 году объем мирового рынка образования увеличится в диапазоне от 7,3 до 7,8 трлн долл. США, а по итогам 2030 года достигнет 10,4 трлн долл. США. Ожидается, что к 2025 году объем международного EdTech-рынка вырастет в диапазоне от 404 до 434 млрд долл. США, а по итогам 2030 года доля международного рынка онлайн-образования должна вырасти до 5,6% (585 млрд долл. США).

1.2 Анализ веб-приложений для изучения иностранных языков

1.2.1 Goodreads

Goodreads – крупнейшая в мире книжная социальная сеть, где есть и русскоязычное сообщество. Пользовательский интерфейс предоставлен на рисунке 1.1. Платформа позволяет отмечать прочитанные книги, меняться произведениями, ставить им оценки и писать рецензии.

В ходе анализа данного веб-приложения были выявлены следующие его достоинства:

- ведение статистики прочитанных книг
- планирование прочтения книг
- выставление оценок книгам
- рецензии

Из недостатков данного приложения можно выделить:

- платформа не адаптирована для изучения иностранных языков.



Рисунок 1.1 – Веб-сайт «Goodreads»

1.2.2 2Books

2Books – современный веб-сайт для чтения книг на английском языке с параллельным переводом текста на русский. Книга в процессе чтения представлена на рисунке 1.2.

Сайт предоставляет следующие возможности:

- чтение иностранных книг с автоматическим переводом;
- выбор книг по жанрам и авторам;

Плюсом приложения является то, что для начинающих изучать язык бесплатно доступны адаптированные английские книги по уровням: Beginner, Elementary, Intermediate, Upper-Intermediate, Advanced. Для более продвинутых – классическая английская литература и современные английские книги в оригинале. На данный момент на сайте имеется 799 книг.

Минусы заключаются в том, что, по незнакомым словам, нельзя создавать карточки и не предоставляется возможность загружать свои книги.

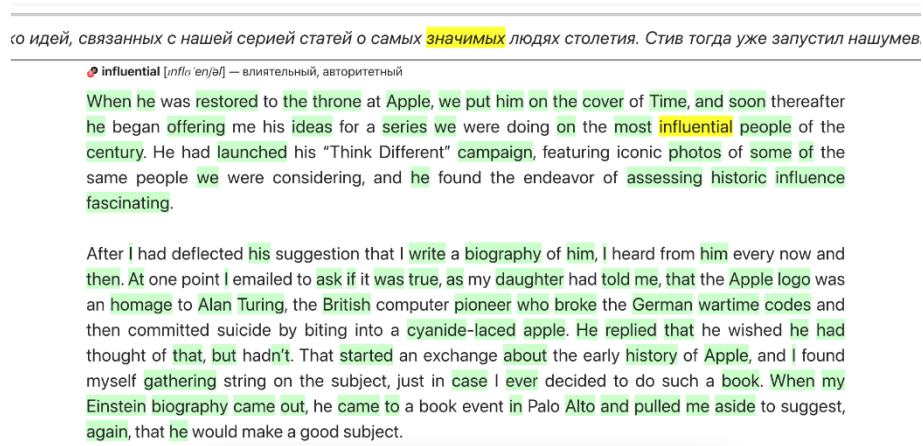


Рисунок 1.2 – Страница чтения на веб-сайте «2Books»

1.2.3 Anki

Anki – одна из самых популярных программ для заучивания разного рода информации с помощью карточек, изображенных на рисунке 1.3. С помощью Anki (в переводе с японского – запоминание) учат не только слова, но и формулы, столицы, определения из учебников, стихи, дорожные знаки – все, что можно запомнить с помощью карточек, но чаще всего Anki используют именно для заучивания слов. В этой программе используется метод интервальных повторений.

Однако приложение не поддерживает чтение книг и создание карточек с переводом.

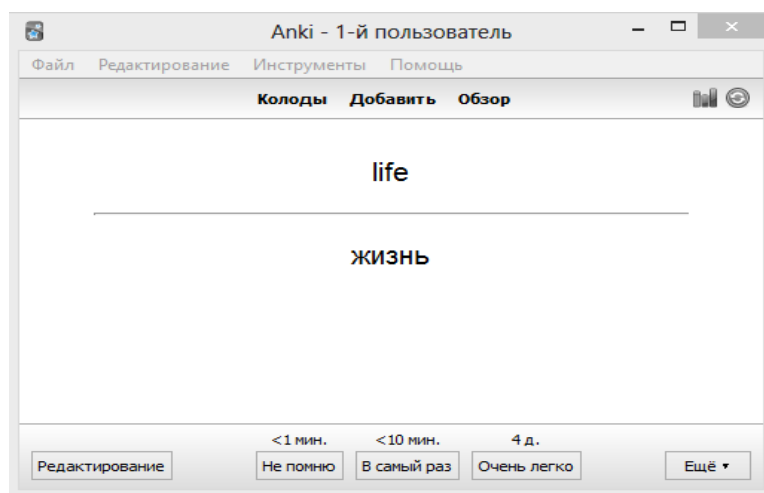


Рисунок 1.3 – Пример карточки в приложении «Anki»

1.2.4 EWA

EWA – приложение для изучения английского языка. Обзор доступных для чтения книг в приложении изображен на рисунке 1.4.

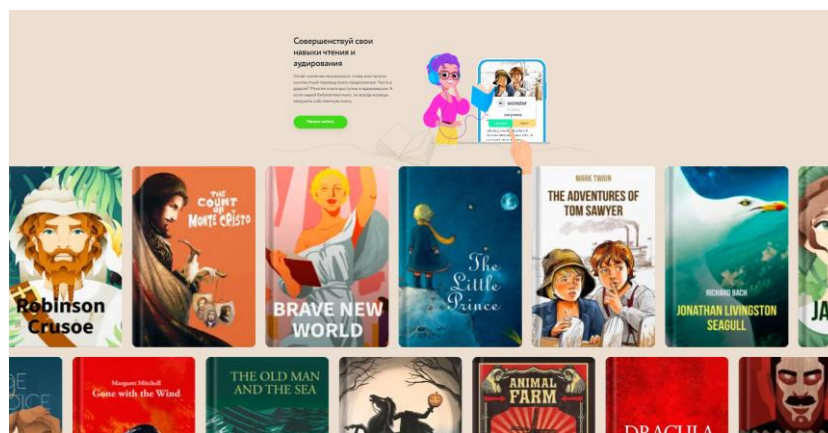


Рисунок 1.4 – Внешний вид приложения «Ewa»

Приложение позволяет каждому пользователю самому решать, как именно ему больше нравится учить английский:

- в EWA доступны для чтения в оригинале больше 1000 книг английских авторов. Книги можно выбирать по уровням – от начального до продвинутого. Также имеется возможность создавать карточки для запоминания;

- разговорные курсы по любимым сериалам и фильмам – безусловный фаворит для изучения английского среди многих пользователей. Курсы разбиты по темам, так что и новички, и обладатели уровня Advanced смогут существенно улучшить свой английский.

Из недостатков следует заметить, что это приложение предназначено только для мобильных устройств или планшетов, а также наличие слишком широкого функционала, который может не пригодится при желании просто читать книгу. Следует упомянуть, что в приложении отсутствует возможность загружать свои книги. Помесячная подписка в приложении стоит 13 долларов США.

1.2.5 Результаты анализа предметной области

В ходе анализа предметной области было установлено, что онлайн-платформа для изучения иностранных языков является востребованным на рынке программным средством.

Проведённый выше анализ предметной области показал, что:

- 1) приложения для изучения иностранных языков сейчас популярны, поскольку знание другого языка дает каждому человеку большое количество новых возможностей;

- 2) веб-приложения с каждым годом всё больше увеличивают охват аудитории, о чем свидетельствует динамика роста в статистике использования данных программ;

- 3) сфера онлайн-образования с каждым годом становится более желанной и предпочтительной;

- 4) большая часть приложений для спорта на данный момент имеет только англоязычную версию. По этой причине такие приложения могут быть не востребованы в нашем регионе;

- 5) на данный момент в Республике Беларусь не существует аналога разрабатываемого приложения;

- 6) приложения, близкие по тематике и содержанию к разрабатываемому программному продукту, имеют ряд недостатков и не могут быть применены для решения проблемы.

Таким образом, наиболее близким аналогом разрабатываемого программного продукта с функциональной точки зрения является приложение EWA. Выявленные в процессе его анализа преимущества и недостатки были учтены при работе над дипломным проектом.

1.3 Эффективность выбранной системы запоминания

При изучении иностранных языков в процессе чтения возникают трудности в понимании из-за недостаточного объема словарного запаса. Отвлекаясь на поиски перевода нужного слова человек теряет концентрацию и внимание столь необходимые для того, чтобы процесс чтения приносил удовольствие и пользу. К сожалению, это не единственная проблема в процессе изучения. Еще одной часто встречающейся проблемой является то, что для того чтобы запомнить информацию мозгу необходимо повторение этой информации не один, а несколько раз. Для этого нужно где-то хранить эти данные и отслеживать периоды времени через которые следует повторять информацию для ее запоминания. Система Лейтнера – система изучения с помощью карточек – решает все эти проблемы. Эта система — простое применение принципа интервальных повторений, где карточки повторяются через увеличивающиеся интервалы.

В этом методе так называемые флеш-карточки рассортированы в группы в зависимости от того, как хорошо ученик усвоил информацию на каждой карточке. Например, при изучении иностранного языка ученик пытается вспомнить значение слова, написанного на флеш-карточке. Если он вспоминает его, то карточка перекладывается в следующую группу. Если же нет, то карточка возвращается в первую группу. Каждая следующая группа повторяется через увеличивающийся интервал. Данный метод может использоваться как для изучения слов иностранного языка, так и запоминания другой информации.

1.4 Требования к проектируемому программному средству

После анализа предметной области, а также обзора уже существующих аналогов необходимо перейти к формированию требования к проектируемой онлайн-платформе.

1.4.1 Назначение проекта

Назначением проекта является разработка онлайн-платформы, предоставляющей возможность изучения иностранных языков.

1.4.2 Основные требования к приложению

Программное средство должно поддерживать следующие основные функции:

- поддержка веб-версии;
- регистрация, аутентификация, авторизация;
- загрузка собственных текстовых материалов;
- категоризация представленных книг;
- управление настройками пользователя;
- управление подпиской;

- создание карточек для неизвестных слов;
- автоматический перевод выбранного слова
- сохранение переведенного слова в виде карточки для запоминания.

1.4.3 Требования к входным данным

Входные данные для программного средства должны быть представлены в виде вводимого пользователем с клавиатуры текста, документов соответствующего формата и содержания и опций, предоставляемых пользовательским интерфейсом приложения.

В бизнес-логике программного средства должны быть реализованы проверки входных данных на корректность, и в случае их невалидности, пользователь должен получать соответствующее уведомление с просьбой ввести данные необходимого формата.

1.4.4 Требования к выходным данным

Выходные данные должны быть представлены посредством отображения информации при помощи различных элементов реализованного и доступного пользовательского интерфейса.

1.4.5 Требования к аппаратному обеспечению клиентской части

Клиентская часть программного средства должна функционировать на ЭВМ со следующими минимальными характеристиками:

- процессор Intel Core с тактовой частотой 2 ГГц и более;
- оперативная память 1 Гб и более;
- возможность выхода в сеть Интернет;

Для корректной работы программного средства необходим один из следующих браузеров с соответствующей минимальной версией:

- Google Chrome 70;
- Opera 58;
- Mozilla Firefox 66;
- Apple Safari 12.0;
- Microsoft Edge 44.

Таким образом, в данной главе были сделаны выводы об актуальности разрабатываемого приложения; выполнен анализ веб-приложений для изучения иностранных языков, в результате которого были выявлены преимущества и недостатки аналогов; прояснен способ запоминания новых слов с помощью создания карточек для запоминания. На основании полученных данных были описаны варианты реализации этапов работы приложения и требования к программному средству. Результаты данного анализа позволили заложить основу для определения технологий, которые необходимо использовать для данной разработки

2 ОБЗОР ИСПОЛЬЗУЕМЫХ В ПРОЕКТЕ ТЕХНОЛОГИЙ

В данном разделе выполняется необходимый на стадии планирования шаг – анализ сведений необходимых для выбора используемых в проекте технологий.

2.1 Выбор платформы приложения

Выбор платформы определяет дальнейшее направление разработки, так как от выбора платформы зависит вид инструментов предоставляемых этой платформой, что в дальнейшем окажет влияние на удобство и скорость реализации программного средства.

2.1.1 Определение подхода к разработке приложения

Обилие различных платформ для создания сайтов создаёт не только благодатную почву для конкуренции, но и усложняет процесс выбора. Чтобы остановиться на самом подходящем решении, часто приходится посвятить много времени изучению возможностей, настройке, тестированию и т.п.

В дополнение к проблеме выбора платформы в современном мире наличие столь обширного списка систем и платформ разработки повлекло за собой появление не менее широкого спектра устройств. Именно поэтому для успешной реализации программного продукта специалистам необходимо спроектировать этот продукт так, чтобы при минимальных изменениях его структуры, архитектуры и исходного кода, приложение продолжало исправно функционировать.

В случаях, когда разработчики проектируют приложение под конкретную систему, они имеют возможность более основательно погрузиться в ее особенности и нюансы, а также погрузиться в происходящее «под капотом» системы, чтобы с минимальными затратами системных ресурсов написать и оптимизировать код. Несмотря на недостаток данного подхода в виде ограничения функционала платформы, можно со справедливостью заметить, что данный подход как никак лучше подходит для реализации небольшого проекта, затратив минимальное количество рабочих ресурсов.

Таким образом, при проектировании онлайн-платформы необходимо учитывать преимущества и недостатки описанного выше подхода к разработке.

2.1.2 Веб-приложение

Веб-приложение – клиент-серверное приложение, в котором взаимодействие клиента с веб-сервисом осуществляется при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется преимущественно, на сервере, обмен информацией происходит по сети. Значимым преимуществом такого подхода можно выделить тот

факт, что клиенты не зависят от конкретной операционной системы пользователя, благодаря этому веб-приложения являются межплатформенными службами.

Веб-приложение состоит из двух частей: клиентской и серверной, тем самым реализуя технологию «клиент-сервер».

Для клиентской части приложения создается пользовательский интерфейс, который генерирует запросы к серверу и обрабатывает ответы от него. Серверная часть получает запросы от клиента, проводит вычисления, формирует веб-страницу и отправляет ее клиенту по сети через протокол HTTP.

Само веб-приложение может выступать в качестве клиента других служб, например, базы данных или другого веб-приложения, расположенного на другом сервере.

В настоящее время существует несколько подходов к разработке веб-приложений: Ajax и WebSocket. При использовании Ajax страницы веб-приложения не перезагружаются целиком, а лишь догружают необходимые данные с сервера, что делает их более интерактивными и производительными. Технология WebSocket не требует постоянных запросов от клиента к серверу и создает двунаправленное соединение, при котором сервер может отправлять данные клиенту без запроса от последнего. Таким образом появляется возможность динамически управлять контентом в режиме реального времени.

В данном проекте для создания веб-приложения на стороне сервера используется технология ASP.NET и язык программирования C#.

2.1.3 MVC

MVC (Model-View-Controller) – архитектурный шаблон, который воплощает в себе три части: модель, представление и контроллер, или, если быть более точными, он делит приложение на три логические части: часть модели, представление, и контроллер. Данный шаблон проектирования используется для разработки мобильных приложений и веб-приложений. MVC управляет всей архитектурой приложений.

Несмотря на то, что часто он подразумевается как шаблон проектирования, мы можем ошибаться если будем называть его только шаблоном проектирования, потому что шаблоны проектирования используются для решения конкретной технической проблемы, в то время, как шаблон архитектуры используется для решения архитектурных задач, поэтому он влияет на всю архитектуру нашего приложения.

Основными причинами выбора MVC стали возможность не повторяться в процессе создания проекта и обеспечение прочной структуры веб-приложения. Данный шаблон состоит из трех компонентов: модель, представление и контроллер.

Модель – уровень известный как самый низкий. Этот уровень отвечает за хранение данных и их логическую обработку. Компонент модели отвечает

за подключение к базе данных, добавление или извлечение данных, также модель отвечает на запросы контроллера, так как контроллер никогда не взаимодействует с базой данных напрямую. Модель в свою очередь никогда не взаимодействует с представлением напрямую.

Представление отвечает за генерирование пользовательского интерфейса для клиента. Поэтому чаще всего компонент представления реализуется через HTML и CSS части. Представления создаются на основе данных, которые собираются компонентом модели, но эти данные не берутся напрямую из модели, а передаются через контроллер, так как представление взаимодействует только с контроллером.

Контроллер – компонент, который обеспечивает взаимосвязь между представлениями и моделью, он действует как посредник. Контроллеру не нужно беспокоиться об обработке логики данных, он просто сообщает модели, что делать. После получения данных из модели он обрабатывает их, а затем берет и отправляет всю эту информацию в представление.

Преимущества выбора MVC шаблона:

- архитектура MVC отделяет пользовательский интерфейс от бизнес-логики;
- компоненты можно использовать повторно;
- простота в обслуживании;
- различные компоненты приложения в MVC могут быть независимо развернуты;
- архитектура позволяет независимо тестировать компоненты.

К недостаткам выбора данного шаблона можно отнести следующие пункты:

- не подходит для больших проектов;
- высокая сложность.

Таким образом, MVC не прост для понимания, что каждый разработчик должен иметь в виду при разработке приложения с его использованием.

MVC – архитектура, которая разделяет ваше программное обеспечение на более мелкие компоненты. Это «разделение» обеспечивает удобочитаемость и модульность, а также облегчает часть тестирования.

2.2 Интегрированная среда разработки

Visual Studio – интегрированная среда разработки (Integrated Development Environment, IDE) от Microsoft, которая предназначена для разработки различных типов программного обеспечения, включая приложения для Windows, веб-приложения, мобильные приложения, игры и многое другое. Помимо редактирования и отладки кода, Visual Studio включает графические конструкторы, компиляторы, средства завершения кода, систему управления

версиями, расширения и многие другие функции для улучшения каждого этапа процесса разработки программного обеспечения.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных.

Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения. Существует множество сторонних плагинов, доступных для скачивания и использования с Visual Studio.

Visual Studio доступен в нескольких версиях, включая бесплатную версию Community и платные версии Professional и Enterprise. Community-версия подходит для большинства небольших и средних проектов, а версии Professional и Enterprise содержат дополнительные функции и инструменты для более сложных проектов и командной разработки.

2.3 Язык программирования C# и платформа .NET

C# – объектно-ориентированный язык программирования, разработанный компанией Microsoft в 1998 году. Язык C# создавался как современная и более безопасная альтернатива языку C++ и был спроектирован для использования в среде разработки .NET Framework. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Язык программирования C# не зависит от платформы и работает с рядом компиляторов и фреймворков, в первую очередь с Microsoft .NET Core и Microsoft .NET Core. Также C# поддерживает управление версиями, чтобы обеспечить совместимость программ и библиотек с течением времени.

Одной из ключевых особенностей C# является его объектно-ориентированный подход. В C# все является объектом, включая базовые типы данных, такие как целые числа и символы. Это позволяет создавать и использовать классы и объекты, что делает код более модульным и повторно используемым.

C# также поддерживает полиморфизм и наследование, что позволяет создавать классы, которые наследуют свойства и методы от других классов. Это

позволяет разработчикам создавать более сложные иерархии классов и повторно использовать код.

Один из главных инструментов С# – сборка мусора. Это означает, что разработчики не должны беспокоиться о выделении и освобождении памяти вручную, так как встроенный механизм сборки мусора автоматически управляет этим процессом.

С# также поддерживает многопоточность, что позволяет разработчикам создавать приложения, которые могут выполнять несколько задач одновременно. Это особенно полезно для приложений, которые должны обрабатывать большие объемы данных или выполнять сложные операции.

В С# также доступны многочисленные стандартные библиотеки и инструменты разработки, которые позволяют создавать более продвинутые приложения. Кроме того, С# является основным языком для разработки Windows-приложений и игр, а также используется для создания приложений на платформе .NET Core для Linux и MacOS.

Программы С# выполняются в .NET, виртуальной системе выполнения, вызывающей общезыковую среду выполнения (CLR) и набор библиотек классов. Среда CLR – реализация общезыковой инфраструктуры языка (CLI), являющейся международным стандартом, от корпорации Майкрософт. CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

Исходный код, созданный на языке С# проходит процесс компилирования в промежуточный язык (IL), который обязательно соответствует спецификациям CLI. Код на языке IL и ресурсы сохраняются в сборке, обычно с расширением .dll. Полученная сборка содержит манифест о своих данных: типах, версии, языке и региональных параметрах.

При выполнении программы С# сборка загружается в среду CLR. Затем среда CLR выполняет JIT-компиляцию из кода на языке IL в инструкции машинного языка. Также среда CLR выполняет другие операции: автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, который выполняет среда CLR, некоторые называют "управляемым кодом". В свою очередь, "неуправляемый код" компилируется на машинный язык, предназначенный для конкретной платформы.

Ключевой особенностью .NET – является обеспечение взаимодействия между языками. Код IL, созданный компилятором С#, соответствует спецификации общих типов (CTS). Код IL, созданный из кода на С#, может взаимодействовать с кодом, созданным из версий .NET для языков F#, Visual Basic, C++. Одна сборка может содержать несколько модулей, написанных на разных языках .NET, и все типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

В дополнение к службам времени выполнения .NET также включает расширенные библиотеки. Эти библиотеки поддерживают множество различных

рабочих нагрузок. Они упорядочены по пространствам имен, которые предоставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа XML, от платформ веб-приложений до элементов управления Windows Forms. Обычно приложения C# активно используют библиотеку классов .NET для решения типовых задач.

.NET Core – новая версия .NET Framework, которая представляет собой бесплатную платформу разработки общего назначения с открытым исходным кодом, выпущенная компанией Microsoft. Это кроссплатформенный фреймворк, который работает в операционных системах Windows, macOS и Linux.

Так как облачные сервера Azure, на которых будет размещен серверный API для приложения, являются продуктом компании Microsoft, использование языка C#, разрабатываемого той же компанией, обеспечит максимальную совместимость и приспособленность инструментов друг к другу.

2.4 ASP.NET

ASP.NET – фреймворк для веб-разработки, разработанный компанией Microsoft. Фреймворк базируется на платформе .NET Framework и использует язык программирования C# или Visual Basic .NET для создания веб-приложений.

ASP.NET включает в себя несколько компонентов, которые помогают разработчикам создавать мощные веб-приложения. Один из главных компонентов – ASP.NET MVC, который представляет собой модель-представление-контроллер (MVC) шаблон проектирования, используемый для создания веб-приложений. Он разделяет приложение на три основные части: модель, представление и контроллер.

Другим важным компонентом является ASP.NET Web API, который позволяет разработчикам создавать RESTful-сервисы для обмена данными между приложениями. Этот компонент обеспечивает легкость разработки и возможность взаимодействия с другими системами. ASP.NET Core – является бесплатным кроссплатформенным фреймворком с открытым исходным кодом для создания облачных приложений, таких как веб-приложения, приложения интернета вещей и мобильные серверные части. Этот фреймворк предназначен для работы как в облаке, так и локально.

Как и .NET Core, его архитектура была модульной с минимальными накладными расходами, а затем другие, более продвинутые функции были добавлены в виде пакетов NuGet в соответствии с требованиями приложения. Это обеспечило высокую производительность, с затратой меньшего объема памяти, а также меньший размер развертывания и простоту обслуживания.

Фреймворк поддерживает несколько языков программирования, таких как C#, Visual Basic .NET, F# и другие. Он также предоставляет библиотеку классов .NET Framework, которая содержит множество полезных функций и

классов, таких как работа с файлами и базами данных, криптография, асинхронное программирование и многие другие.

ASP.NET также обладает сильной защитой, предоставляя разработчикам множество инструментов для обеспечения безопасности веб-приложений. В него включены средства защиты от атак вроде XSS, CSRF и инъекций SQL.

В целом, ASP.NET является мощным и гибким фреймворком для создания веб-приложений. Он предоставляет множество компонентов и инструментов, которые помогают разработчикам создавать высококачественные приложения быстрее и эффективнее.

2.5 PostgreSQL

PostgreSQL – объектно-реляционная система управления базами данных (ORDBMS), которая разработана для обеспечения безопасного хранения, масштабирования и управления самыми сложными рабочими нагрузками данных. Она использует и расширяет язык SQL в сочетании со многими функциями, что делает ее одной из самых мощных систем управления базами данных с открытым исходным кодом.

PostgreSQL имеет богатый список функций, которые обеспечивают производительность и безопасность, а также связаны с программными расширениями и конфигурацией. Среди таких функций можно выделить поддержку объектно-ориентированных возможностей, полнотекстовый поиск, географическую индексацию, поддержку многопользовательской работы и распределенной работы.

PostgreSQL была создана в 1986 году в рамках проекта POSTGRES в Калифорнийском университете в Беркли и уже более 30 лет активно развивается на базовой платформе. Она также обладает высокой степенью совместимости с другими СУБД, такими как Oracle, MySQL и Microsoft SQL Server, что позволяет легко мигрировать существующие приложения на PostgreSQL.

PostgreSQL поддерживает многие операционные системы, включая Windows, macOS и различные дистрибутивы Linux. Она также имеет широкое сообщество пользователей и разработчиков, которые предоставляют богатый набор ресурсов, таких как документация, форумы и репозитории, для поддержки пользователей и улучшения системы.

Одной из основных причин выбора использования PostgreSQL является её большой список функций, которые призваны обеспечить производительность и безопасность, а также связаны с программными расширениями и конфигурацией.

PostgreSQL технически является объектно-реляционной системой управления базами данных (ORDBMS). Это означает, что она имеет те же реляционные возможности, что и СУБД, но в дополнение имеет некоторые объектно-ориентированные функции.

Эти функции являются мощными инструментами, помогающие

работать с данными и базами данных, используя некоторые из тех же методов, что и при программировании. Повышенная гибкость позволяет моделировать различные типы и отношения внутри системы баз данных, а не программно. Это может помочь сохранить согласованность и адаптировать предполагаемое поведение к фактическим данным.

2.6 Entity Framework (EF) Core

Entity Framework Core (EF Core) – ORM (Object-Relational Mapping) фреймворк для .NET, разработанный Microsoft, который предоставляет разработчикам инструменты для работы с базами данных и объектами в .NET приложениях. EF Core позволяет работать с различными базами данных, включая SQL Server, PostgreSQL, MySQL, SQLite и другие.

Основным принципом EF Core является сопоставление объектно-ориентированных моделей с схемой базы данных. То есть, вместо написания SQL-запросов для доступа к данным в базе данных, разработчики могут работать с объектами .NET, которые затем преобразуются в соответствующие SQL-запросы. Это позволяет более эффективно работать с данными и уменьшает объем написания рутины для доступа к данным.

EF Core использует линейный подход к миграции баз данных, что позволяет разработчикам изменять схему базы данных в пакете миграции, который затем применяется к базе данных при запуске приложения. Это помогает разработчикам легко обновлять базы данных и поддерживать их в актуальном состоянии.

EF Core также предоставляет множество инструментов для работы с данными, включая LINQ to Entities, которая позволяет написать LINQ-запросы для доступа к данным в базе данных, а также инструменты для работы с транзакциями и кэшем. Кроме того, EF Core предоставляет возможность работать с несколькими источниками данных одновременно, что может быть полезно при работе с распределенными приложениями.

В целом, Entity Framework Core является мощным инструментом для работы с базами данных в .NET приложениях, предоставляющим множество инструментов для работы с данными и схемой базы данных, а также поддерживающим множество различных источников данных.

EF Core может служить объектно-реляционным картографом, который:

- позволяет разработчикам .NET работать с базой данных, используя .NET объекты;
- устраняет необходимость в большей части кода доступа к данным, который обычно требуется написать;
- EF Core поддерживает множество ядер баз данных.

2.7 JavaScript

JavaScript – многопарадигмальный динамический язык с типами и операторами, стандартными встроенными объектами и методами. Его синтаксис основан на языках Java и C, поэтому многие структуры из этих языков применимы и к JavaScript. JavaScript поддерживает объектно-ориентированное программирование с использованием прототипов объектов и классов. Он также поддерживает функциональное программирование, поскольку функции являются первоклассными, их можно легко создавать с помощью выражений и передавать по кругу, как любой другой объект.

JavaScript – язык, который может быть использован для создания интерактивных веб-страниц, динамических пользовательских интерфейсов и серверных приложений. Он имеет множество встроенных функций и методов, что позволяет разработчикам создавать богатые и мощные веб-приложения. JavaScript также имеет возможность работы с другими языками, такими как HTML и CSS, что позволяет создавать динамические и интерактивные веб-страницы.

JavaScript можно использовать как на стороне клиента (в браузере), так и на стороне сервера (в Node.js). Он имеет множество библиотек и фреймворков, таких как jQuery, React и Angular, которые упрощают создание интерактивных веб-приложений. JavaScript сначала был известен как LiveScript, но Netscape изменил свое название на JavaScript, возможно, из-за ажиотажа, вызванного Java. JavaScript впервые появился в Netscape 2.0 в 1995 году под названием LiveScript. Универсальное ядро языка было встроено в Netscape, Internet Explorer и другие веб-браузеры.

Клиентский JavaScript является наиболее распространенной формой языка. Скрипт должен быть включен в HTML-документ или на него должна быть ссылка, чтобы код мог быть интерпретирован браузером.

Это означает, что веб-страница не обязательно должна быть статическим HTML, но может включать программы, которые взаимодействуют с пользователем, управляют браузером и динамически создают HTML-контент.

Механизм JavaScript на стороне клиента предоставляет множество преимуществ по сравнению с традиционными CGI-скриптами на стороне сервера. Например, можно использовать JavaScript, чтобы проверить, ввел ли пользователь действительный адрес электронной почты в поле формы. Код JavaScript выполняется, когда пользователь отправляет форму, и только в том случае, если все записи действительны, они будут отправлены на веб-сервер.

JavaScript также имеет много возможностей для обработки событий, что позволяет реагировать на пользовательские действия, такие как нажатие кнопки или ввод данных в форму. Он также имеет возможность работы с асинхронным кодом, что делает его очень мощным для создания веб-приложений, которые могут обращаться к серверу без перезагрузки страницы.

JavaScript является одним из наиболее популярных языков программирования в мире и постоянно развивается, поэтому существует множество ресурсов для обучения и поддержки, включая множество библиотек и фреймворков, сообщества разработчиков и документацию.

2.8 React

React – библиотека JavaScript, используемая для создания повторно используемых компонентов пользовательского интерфейса. Согласно официальной документации React – библиотека для создания составных пользовательских интерфейсов. Библиотека поощряет создание повторно используемых компонентов пользовательского интерфейса, которые представляют данные, изменяющиеся с течением времени.

React используют в качестве V в MVC. React абстрагирует разработчика от DOM, предлагая более простую модель программирования и более высокую производительность. React также может выполнять рендеринг на сервере с помощью Node, и он может запускать собственные приложения с помощью React Native. React реализует односторонний реактивный поток данных, который сокращает количество шаблонов и о котором легче рассуждать, чем о традиционной привязке данных.

Основная идея React заключается в том, чтобы разбить UI на отдельные компоненты и обновлять только те компоненты, которые действительно нуждаются в изменении. В React для этого используется виртуальная DOM (DOM – Document Object Model). Виртуальная DOM – это копия реальной DOM, которая находится в памяти, и работать с ней гораздо быстрее, чем с реальной DOM. Когда изменяется состояние компонента, React сравнивает виртуальную DOM с реальной и обновляет только те элементы, которые изменились.

React не является полноценным фреймворком, как Angular или Vue.js. Вместо этого он может использоваться с другими библиотеками и фреймворками, такими как Redux, MobX, Next.js и Gatsby.

React также обладает расширяемой системой компонентов, что означает, что разработчики могут создавать свои собственные компоненты и использовать их в своих приложениях.

React поддерживает использование JSX, это специальный синтаксис, который позволяет использовать HTML-подобные теги в JavaScript коде, что делает его более читаемым и понятным.

Из-за своего компонентного подхода и эффективной системы обновления виртуальной DOM, React позволяет создавать сложные пользовательские интерфейсы с большой скоростью и эффективностью, делая его популярным инструментом для разработки веб-приложений.

React обладает следующими особенностями:

- Компоненты. React – все, что связано с компонентами. Вам нужно думать обо всем как о компоненте. Это поможет вам поддерживать код при работе над крупномасштабными проектами.

- Однонаправленный поток данных и Flux – React реализует односторонний поток данных, что позволяет легко рассуждать о вашем приложении. Поток – шаблон, который помогает сохранять ваши данные однонаправленными.

- Лицензия – React лицензирован компанией Facebook Inc. Документация лицензирована по лицензии CC BY 4.0.

Главные преимущества React:

- Использование виртуального DOM, который является объектом JavaScript. Это повышает производительность приложений, поскольку виртуальный DOM на JavaScript работает быстрее обычного DOM.

- Может быть использован на стороне клиента и сервера, а также с другими фреймворками.

- Шаблоны компонентов и данных улучшают удобочитаемость, что помогает поддерживать большие приложения.

Однако библиотека React имеет некоторые ограничения:

- Охватывает только уровень просмотра приложения, следовательно, вам все равно нужно выбрать другие технологии, чтобы получить полный набор инструментов для разработки.

- Использует встроенные шаблоны и JSX, что может показаться неудобным некоторым разработчикам.

2.9 Google Translate API

API (Application Programming Interface) – интерфейс, который позволяет программам и системам взаимодействовать друг с другом и обмениваться информацией. API может быть представлен как набор функций, методов и протоколов, которые используются для создания программного обеспечения.

API может быть, как внутренним, то есть использоваться внутри компании для обмена информацией между различными приложениями, так и внешним, то есть предоставлять доступ к определенным данным и функциям через сеть интернет для сторонних приложений и разработчиков.

API может использоваться в различных областях, включая веб-разработку, мобильную разработку, анализ данных и другие. Одним из наиболее популярных видов API является REST API (Representational State Transfer), который основан на протоколе HTTP и позволяет передавать данные в формате JSON или XML.

Одной из основных целей использования API является упрощение и ускорение разработки программного обеспечения путем использования готовых функций и методов, которые уже реализованы и протестированы. Кроме

того, API может улучшить взаимодействие между различными приложениями и системами, позволяя им обмениваться информацией и работать более эффективно.

Чтобы использовать API, разработчики должны сначала получить доступ к нему, который может быть ограничен различными параметрами, такими как ключ доступа, учетная запись пользователя или другие методы авторизации. Затем разработчики могут использовать документацию API, чтобы понять, какие методы доступны и как использовать их для своих приложений.

API играет важную роль в современной разработке программного обеспечения и является необходимым инструментом для многих разработчиков, которые стремятся создать эффективные и функциональные приложения. Google Translate API позволяет веб-сайтам и приложениям динамически переводить текст программно через API. Google Translate использует для перевода текста предварительно подготовленную Google или пользовательскую модель машинного обучения. По умолчанию облачный перевод использует модель Google pre-trained Neural Machine Translation (NMT), которую Google обновляет с полурегулярной частотой, когда становится доступно больше обучающих данных или более совершенных методов.

Главными достоинствами Google Translate API считаются:

- Непревзойденная языковая поддержка. Машинный перевод можно использовать для распознавания более чем ста языков, от африкаанса до зулу.
- Лучшее в своем классе качество. Google имеет богатую историю предоставления услуг перевода потребителям и организациям. Проверенные модели и инструменты обеспечивают переводческий опыт Google с высочайшей в отрасли точностью.
- Специфика предметной области. Услуги перевода можно настроить таким образом, чтобы они понимали отраслевой сленг или термины, относящиеся к конкретной предметной области. Также можно сохранять контекст и смысл при переводах технических документов, описаний продуктов и социального контента.

2.10 Azure Cloud и Azure Blob

Microsoft Azure – общедоступная гибкая облачная платформа. Она предоставляет ряд облачных услуг, включая вычисления, аналитику, работу с хранилищем и сетью. Azure дает разработчикам неограниченную свободу для создания, развертывания приложений и управления ими в глобальной сети с меньшими затратами и с большей безопасностью, по сравнению с большинством локальных решений.

Поскольку Microsoft Azure состоит из множества сервисных предложений для разработки и масштабирования новых приложений или запуска существующих приложений в общедоступном облаке, варианты ее

использования чрезвычайно разнообразны. Запуск виртуальных машин или 36 контейнеров в облаке — одно из самых популярных применений Microsoft Azure.

Также Azure широко используется в качестве платформы для размещения баз данных в облаке. Microsoft предлагает бессерверные реляционные базы данных, такие как Azure SQL, и нереляционные базы данных, такие как NoSQL.

Хранилище BLOB-объектов Azure – решение Microsoft для хранения объектов в облаке. Хранилище BLOB-объектов оптимизировано для хранения больших объемов неструктурированных данных.

Неструктурированные данные – данные, которые не соответствуют определенной модели данных или определению, например текстовые данные или картинки.

Пользователи или клиентские приложения могут получать доступ к объектам в хранилище Azure Blob через HTTP/HTTPS из любой точки мира.

Клиентские библиотеки доступны для разных языков, в том числе и для .NET.

Благодаря нескольким уровням хранения и автоматизированному управлению жизненным циклом Azure Blob позволяет экономично хранить огромные объемы неструктурированных данных, что делает данную платформу подходящим решением для разработки данного приложения.

2.11 Firebase

Firebase – облачная платформа, разработанная Google, для создания мобильных и веб-приложений. Первоначально она представляла собой базу данных реального, теперь же она имеет 18 сервисов и специальный интерфейс.

Вся платформа представляет собой решение Backend-as-a-Service, которое включает в себя службы для создания, тестирования и управления приложениями. Решения BaaS позволяют подключить бэкенд к приложению через специальные API для каждой отдельной службы. Список платформ, с которыми интегрируется Firebase, включает Android, iOS, Web и Unity.

В начале работы Firebase абсолютно бесплатный и в процессе разработки не требует оплаты большинства своих услуг. Как только будет достигнут определенный объема базы данных или понадобится конкретная услуга, можно перейти на один из планов оплаты.

Для данного приложения не требуются продвинутое функции, поэтому Firebase является идеальным бесплатным решением.

2.11.1 Firebase Authentication

Большинство приложений имеют функцию аутентификации, которая необходима для идентификации пользователя. Firebase Authentication

стремится упростить создание безопасных систем аутентификации, улучшая при этом процесс входа и первоначальное впечатление от использования приложения для конечных пользователей. Он предоставляет комплексное решение для идентификации, поддерживающее вход с помощью электронной почты и пароля, аутентификацию по телефону, а также вход с помощью учётной записи Google, Twitter, Facebook и GitHub.

Созданная той же командой, которая разработала Google Sign-in, Smart Lock и Chrome Password Manager, Firebase Security использует внутренний опыт Google по управлению одной из крупнейших баз данных учетных записей в мире.

На настройку собственной системы аутентификации могут уйти месяцы, и для ее обслуживания в будущем потребуется команда инженеров.

Firebase Authentication позволяет настроить всю систему аутентификации приложения, прилагая минимум усилий и времени для написания кода.

Также Firebase Authentication позволяет связать учетные записи, так что пользователь в итоге будет иметь один профиль, независимо от того, войдет ли он в Facebook сегодня или в Google завтра.

Одним из плюсов данного решения является возможность использования различных продуктов Firebase вместе. Например, Cloud

Firestore интегрируется с Firebase Authentication, чтобы обеспечить прямой доступ к базе данных для клиента, сохраняя при этом безопасность на стороне сервера.

2.11.2 Firebase Analytics

Хотя Google Analytics – бесплатный инструмент, предоставляемый Google, его интеграция становится незаметной при работе с Firebase. Google Analytics совместим с iOS, Android, Web, C++ и Unity. Это бесплатное решение для аналитики сообщает разработчикам, как пользователи относятся к их мобильному или веб-приложению. Firebase Analytics также полезен для увеличения показателей удержания и вовлеченности клиентов, что делает его незаменимым инструментом для определения векторов для улучшения приложения. Также имеется возможность создания отчетов в неограниченном количестве и абсолютно бесплатно.

SDK автоматически фиксирует ряд событий и свойств, а также позволяет определять свои собственные настраиваемые события для измерения показателей, которые имеют особое значение для бизнеса. После получения данных они доступны на панели инструментов через консоль Firebase. Эта информационная панель предоставляет подробные — от сводных данных, таких как активные пользователи и демографические данные, до более подробных данных, таких как определение наиболее покупаемых товаров.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В рамках данного дипломного проекта было разработано программное средство, состоящее из двух обособленных модулей: серверного API, обеспечивающего работу с базой данных, и React-приложения. Рассмотрим архитектуру данных модулей и связанных с ними компонентов.

Первым шагом, который необходимо сделать, при разработке программного средства – считается выбор архитектурного шаблона. Хорошо продуманная архитектура программного средства является ключевым фактором успеха проекта. Существует несколько причин, по которым важно тщательно продумать архитектуру программного средства:

- хорошо спроектированная архитектура обеспечивает масштабируемость приложения. Она позволяет легко добавлять новые функции и улучшать производительность, не нарушая стабильность и работоспособность приложения;
- при продуманной архитектуре код становится более читаемым и понятным для других разработчиков, что упрощает поддержку и развитие приложения;
- хорошо спроектированная архитектура обеспечивает надежность приложения. Она предусматривает резервирование и обработку ошибок, что позволяет избежать критических ошибок и сбоев в работе приложения;
- тщательно спроектированная архитектура позволяет достигать высокой эффективности приложения. Это происходит благодаря использованию эффективных алгоритмов и структур данных, которые позволяют быстро обрабатывать большие объемы данных;
- хорошо спроектированная архитектура делает приложение более удобным в использовании. Это происходит благодаря удобному интерфейсу и простоте использования, что делает приложение более привлекательным для пользователей.

В целом, продуманная архитектура программного средства является необходимым условием для создания успешного и конкурентоспособного продукта, который удовлетворяет потребности пользователей и соответствует требованиям рынка. Поэтому важно уделить достаточно времени и усилий для разработки эффективной и масштабируемой архитектуры приложения.

3.1 Формирование функциональных требований

Сайт должен позволять пользователям загружать текстовые материалы, приобретать и читать книги; должна быть возможность создания и сохранения карточек для запоминания неизвестных слов/выражений, иностранных слов с автоматическим переводом

- Сайт должен предоставлять:
- регистрацию;
 - авторизацию;
 - загрузку собственных текстовых материалов;
 - создание карточек для неизвестных слов;
 - автоматический перевод выделенного текста на иностранном языке и также сохранение его в виде карточки;
 - настройка аккаунта пользователя.

Основные возможности пользователя отображены на диаграмме вариантов использования программного средства на рисунке 3.2.

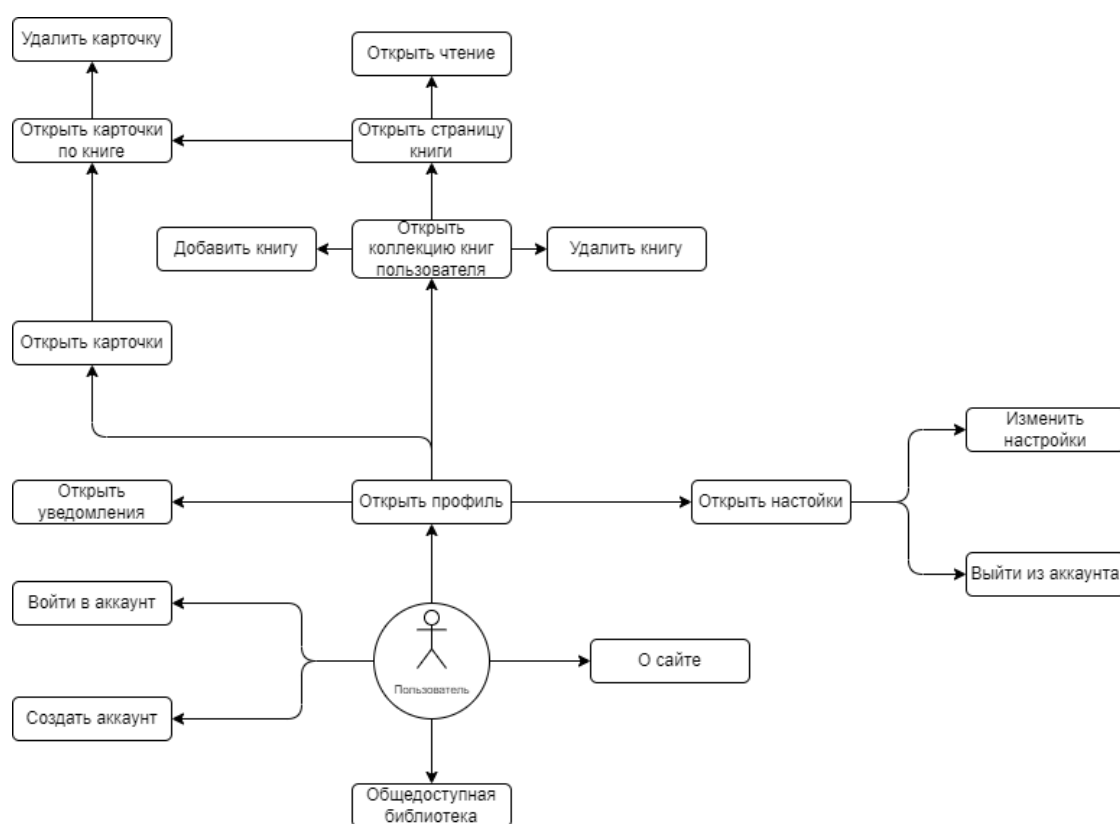


Рисунок 3.1 – Диаграмма вариантов использования ПС

3.2 Model-View-Controller (MVC) шаблон

Model-View-Controller – популярный архитектурный шаблон, который разделяет приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Каждый из этих компонентов имеет свою ответственность и выполняет определенную роль в приложении.

MVC является широко используемым и распространенным шаблоном в различных технологических стеках и языках программирования. Он способствует удобству разработки, поддержке и масштабируемости приложений, а

также способствует разделению ответственности между различными компонентами.

Рассмотрим реализацию подхода MVC (Model-View-Controller) в проекте, где ASP.NET используется для бизнес-логики и доступа к данным, а React – для слоя отображения. Вот общая схема реализации MVC в данном контексте:

- в ASP.NET создается модель, которая представляет данные и бизнес-логику нашего программного средства. Модель содержит классы, структуры или объекты, которые представляют сущности приложения и их взаимодействие. Модель также может включать классы для доступа к данным, такие как Entity Framework, которые обеспечивают связь с базой данных;

- в ASP.NET создаются контроллеры, которые будут обрабатывать запросы от клиента и управлять бизнес-логикой. Контроллеры могут содержать методы действий (action methods), которые принимают входные данные, взаимодействуют с моделью и возвращают результаты в виде данных или представлений. Контроллеры взаимодействуют с моделью, вызывая методы модели для выполнения операций, таких как чтение, создание, обновление или удаление данных.

- в React создаются компоненты представления, которые отвечают за отображение данных и взаимодействие с пользователем. Компоненты представления получают данные из контроллеров или из промежуточного состояния (например, Redux-хранилище) и используют их для отображения пользовательского интерфейса. Компоненты представления также отправляют пользовательские действия обратно в контроллеры, чтобы инициировать изменение состояния приложения.

Взаимодействие между React-представлениями и ASP.NET-контроллерами осуществляется посредством использования API, предоставляемого ASP.NET.

React-компоненты отправляют запросы на сервер, передавая параметры и данные, а контроллеры обрабатывают эти запросы и возвращают соответствующие данные или представления в ответ.

Важно отметить, что в данном подходе React не является полноценным "представлением" в классическом смысле MVC, так как взаимодействие с пользователем осуществляется через JavaScript и интерфейс пользователя основан на компонентах React. Однако, он выполняет функцию отображения данных и пользовательского интерфейса, а контроллеры ASP.NET обрабатывают бизнес-логику и взаимодействуют с моделью.

3.3 React-приложение

В ходе разработки данной онлайн-платформы для изучения иностранных языков было принято решение для создания слоя отображения использовать React-приложение.

Внутри React-приложения можно использовать различные паттерны и подходы для управления состоянием и организации кода в зависимости от сложности проекта и требований. Перечислим используемые в приложении паттерны:

- Компонентный подход: React по своей природе поддерживает компонентный подход, где можно разбить пользовательский интерфейс на множество маленьких и переиспользуемых компонентов. Это помогает создавать модульный и легко поддерживаемый код.

- Redux – библиотека для управления состоянием в приложениях React. Она реализует паттерн Flux и предоставляет единое хранилище состояния для всего приложения. Redux удобен для приложений с большим объемом данных и сложной логикой.

В конечном итоге, выбор использования этих подходов зависит от сложности проекта и команды разработчиков. Они могут быть взаимосвязаны и дополнять друг друга, чтобы создать хорошо организованное, модульное и масштабируемое приложение.

В ходе реализации данного дипломного проекта за основу был взят архитектурный паттерн Flux в совокупности с подходом Feature-Sliced Design (дизайн по срезам функциональности), а для управления состоянием приложения была выбрана библиотека Redux.

3.3.1 Flux

В React-проектах часто применяется одна основная архитектура, которая обычно называется "однонаправленный поток данных" или "фронтенд-архитектура Flux", схематично изображенная на рисунке 3.2. Эта архитектура включает в себя использование состояний (state) и свойств (props) для передачи данных и управления состоянием компонентов.

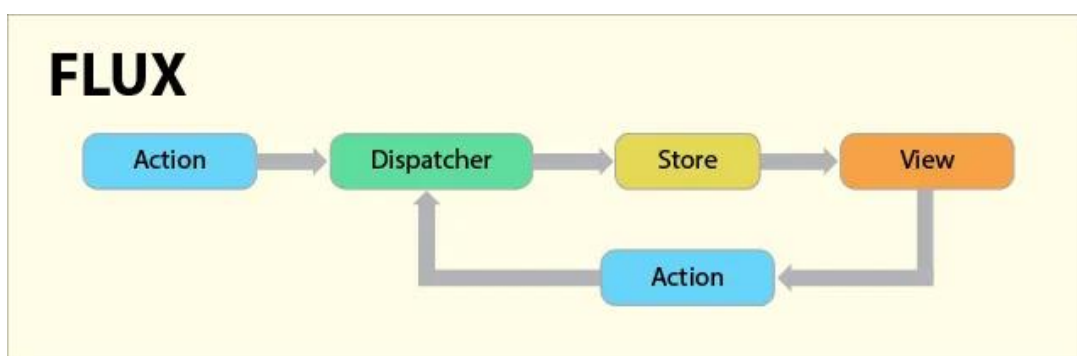


Рисунок 3.2 – Шаблон проектирования Flux

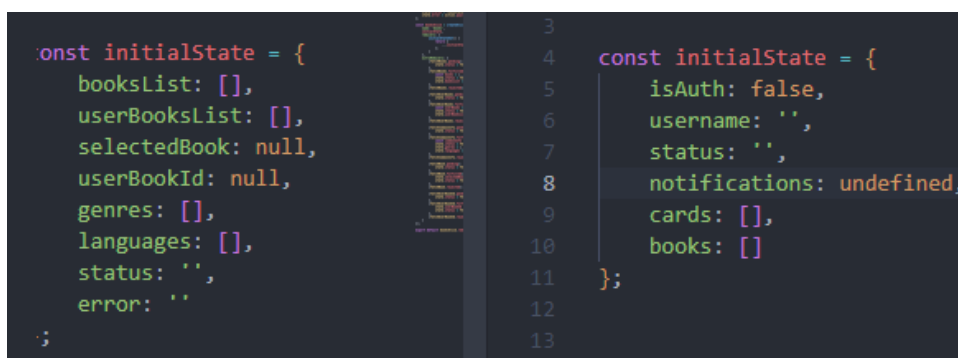
Flux – архитектурный паттерн, который используется для управления состоянием в React-приложениях. Он был разработан компанией Facebook и стал популярным в сообществе React-разработчиков. Flux предлагает однонаправленный поток данных и разделяет данные и логику приложения на четыре

основных компонента: действия (actions), диспетчер (dispatcher), хранилище (store) и представление (view).

Действия представляют собой простые объекты, которые описывают события или операции, происходящие в приложении. Они определяются как константы и могут иметь некоторые данные, необходимые для выполнения операции. Например, у вас может быть действие "Добавить элемент", которое содержит данные о добавляемом элементе. Действия создаются и вызываются в представлении или в ответ на взаимодействие пользователя.

Диспетчер – централизованный узел, который принимает действия и направляет их к соответствующему хранилищу. Он является своего рода ретранслятором действий, не обрабатывая их самостоятельно. Диспетчер также обеспечивает уникальность потока действий, гарантируя, что одновременно будет обрабатываться только одно действие.

Хранилище содержит состояние приложения и логику его изменения в ответ на действия. Оно является одним источником правды и хранит данные в одном месте. Изначальный вид хранилища продемонстрирован на рисунке 3.3. Хранилище обрабатывает действия, определяет, какие данные нужно изменить и обновляет свое состояние соответствующим образом. После обновления хранилище уведомляет представление о новых данных путем отправки уведомления. В данном приложении хранилище состоит из двух основных частей: bookReducer и userReducer. Разделение хранилища на разные части в зависимости от используемого функционала позволяет достигать большей гибкости и осмысленности в приложении.



```
const initialState = {
  booksList: [],
  userBooksList: [],
  selectedBook: null,
  userBookId: null,
  genres: [],
  languages: [],
  status: '',
  error: ''
};

const initialState = {
  isAuthenticated: false,
  username: '',
  status: '',
  notifications: undefined,
  cards: [],
  books: []
};
```

Рисунок 3.3 – Изначальное состояние хранилища

Представление отображает данные из хранилища и реагирует на изменения состояния, обновляя себя соответственно. Когда представление отправляет действия, они переходят через диспетчера и попадают в хранилище для обработки. Представление также подписывается на хранилище, чтобы получать уведомления.

3.3.2 Redux

Redux – популярная библиотека управления состоянием, которая широко используется в React-приложениях. Она предлагает предсказуемый однонаправленный поток данных и помогает управлять состоянием приложения централизованно.

К важным составляющим Redux, содержащихся в данном приложении можно отнести:

- однонаправленный поток данных: Redux реализует паттерн Flux и предлагает однонаправленный поток данных, который облегчает отслеживание и управление состоянием приложения. Данные в Redux движутся только в одном направлении, что обеспечивает предсказуемость и прозрачность изменений состояния;

- централизованное хранилище: Redux использует централизованное хранилище состояния, которое является единственным источником правды для всего приложения. Состояние хранится в одном объекте, называемом "store", изображенном на рисунке 3.5. Это упрощает отслеживание и обновление состояния приложения;

- действия (actions): действия в Redux представляют собой простые объекты, которые описывают события или операции, происходящие в приложении. Они определяются как константы и могут иметь полезную нагрузку данных. Действия инициируются компонентами и передаются в Redux для обработки.

- редукторы (reducers): редукторы в Redux являются чистыми функциями, которые обрабатывают действия и возвращают новое состояние приложения. В контексте React и функционального программирования, «чистая функция» это функция которая соответствует двум основным правилам: изолированности и отсутствию побочных эффектов. Редукторы определяют, как состояние изменяется в ответ на действия. Каждый редуктор отвечает за определенную часть состояния.

- подписка на состояние: компоненты представления могут подписываться на состояние из Redux-хранилища и получать только необходимые им данные. При изменении состояния, Redux автоматически уведомляет подписанные компоненты о новых данных, что позволяет им обновиться;

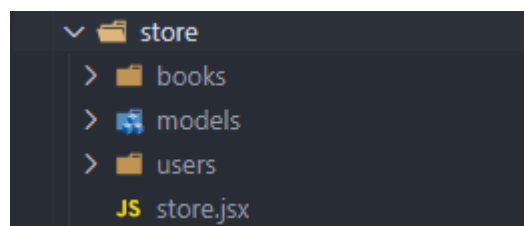


Рисунок 3.4 – Централизованное хранилище

В качестве действий в Redux может выступать `thunk` функция. `Thunk` является средством для обработки асинхронных действий в Redux. Он представляет собой функцию, которая оборачивает действие (action) и позволяет выполнять асинхронные операции, такие как запросы к серверу, перед отправкой результата в Redux хранилище.

Основные принципы `thunk` в Redux:

- позволяет создавать действия в виде функций, а не только в виде объектов, как это делается в стандартном Redux. Это дает возможность выполнять асинхронные операции внутри действия;

- функция `thunk` принимает два параметра – `dispatch` и `getState`. `Dispatch` используется для отправки других действий в Redux, а `getState` позволяет получить текущее состояние хранилища, пример вызова `dispatch` функции изображен на рисунке 3.5;

- `thunk` дает возможность задерживать отправку действия в Redux до завершения асинхронной операции. Это позволяет выполнять дополнительные действия перед отправкой результата в хранилище или делать условные проверки;

- вместо создания обычного действия, создается функция, которая принимает параметры и возвращает функцию с доступом к `dispatch` и `getState`. Внутри этой функции можно выполнять асинхронные операции, такие как запросы к API, пример такого запроса изображен на рисунке 3.6, и в конечном итоге отправлять другие действия в Redux.

```
useEffect(() => {  
  if (userBooks.length === 0) {  
    dispatch(fetchUserBooks());  
  }  
  
  if (genres.length === 0) {  
    dispatch(fetchCommonInfo());  
  }  
}, [dispatch]);
```

Рисунок 3.5 – Вызов `dispatch` функции

```
const fetchBooks = createAsyncThunk(
  'books/FetchBooks',
  async (input, { rejectWithValue }) => {
    try {
      const url = urlHelper.getUrlByTemplate(
        serviceUrls.getBooks,
      );
      const data = await requestHelper.get(url);

      return [data, input];
    } catch (error) {
      return rejectWithValue(error.message);
    }
  }
);
```

Рисунок 3.6 – Пример использования thunk в Redux

3.3.3 Feature-Sliced Design

Feature-Sliced Design (FSD) – паттерн архитектуры, который предлагает организацию кода в React-приложениях на основе функциональных возможностей или модулей.

Такой подход обладает следующими плюсами:

- FSD ставит акцент на модульности, разбивая приложение на отдельные возможности или модули. Каждая возможность имеет свою собственную структуру папок, содержащую компоненты, стили, логику и тесты. Это позволяет создавать независимые компоненты, которые можно использовать не один раз, улучшая поддерживаемость и расширяемость кода;

- FSD способствует явному разделению ответственностей между различными частями приложения. Каждая возможность обычно отвечает за определенный функциональный аспект приложения, что упрощает понимание и изменение кода;

- структура Feature-Sliced Design делает код более организованным и понятным, что обеспечивает лучшую читаемость и обслуживаемость. Легко найти нужные компоненты и связанный с ними код. Это особенно полезно при работе в команде, где разработчики могут легко ориентироваться в коде других членов команды;

Объединение Flux и Feature-Sliced Design в React-приложении принести преимущества в организации кода и управлении состоянием. Для этого эти два подхода были соединены следующим образом:

- осуществилось разделение приложения на логические срезы (features) по страницам, используемым в приложении (user-page, book-page и др.). Каждый срез содержит свои компоненты, утилиты и стили, относящиеся к данной функциональности. Срезы изображены на рисунке 3.7. Это позволяет легко

навигировать по коду, улучшает его читаемость и облегчает поддержку и масштабирование приложения;

- создано централизованное хранилище состояния, используя принципы Flux-архитектуры. Хранилище содержит состояние, которое общее для всего приложения. Оно служит единственным источником данных для всех срезов функциональности;

- были введены действия (actions) и диспетчер (dispatcher) для передачи данных и управления потоком информации между срезами функциональности и централизованным хранилищем. Действия были определены на уровне среза функциональности и содержат данные, специфичные для этого среза. Диспетчер реализован в виде глобального сервиса, который принимает действия от различных срезов и направляет их в хранилище для обработки;

- компоненты представления в каждом срезе функциональности подписаны на состояние, необходимое для их работы, из централизованного хранилища. Подписка на состояние обеспечивает обновление компонентов при изменении данных в хранилище;

- когда компоненты представления инициируют действия, они вызывают соответствующие действия, которые попадают в диспетчер. Диспетчер передает действия в хранилище, которое обновляет состояние в соответствии с данными действия.

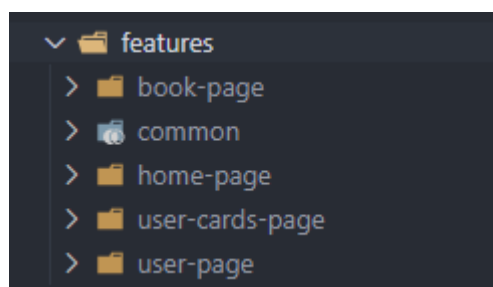


Рисунок 3.7 – Структура функциональных модулей

3.3.4 Слой отображения

Слой отображения (View) в архитектуре MVC представляет собой компонент, который отвечает за отображение данных пользователю и обработку пользовательского ввода. Он является интерфейсом между пользователем и остальными компонентами архитектуры.

Основными функциями слоя отображения в MVC являются:

- отображение данных;
- обработка пользовательского ввода;
- рендеринг и обновление интерфейса;
- взаимодействие с другими компонентами;
- разделение отображения и логики.

В React-приложении слой отображения обычно реализуется с помощью компонентов. Компоненты в React являются строительными блоками пользовательского интерфейса и отвечают за отображение данных и управление пользовательским вводом. React-приложение может реализовывать слой отображения при помощи функциональных компонентов.

Функциональные компоненты в React – один из способов создания компонентов в React с использованием функций. Они являются более простым и прямолинейным подходом к созданию компонентов без использования классов. Функциональные компоненты представляют собой простые функции, которые принимают объект props в качестве аргумента и возвращают JSX (JavaScript XML) для отображения. Функциональные компоненты обычно более просты и читаемы, поскольку они имеют меньше синтаксического шума и меньше "лишнего" кода, связанного с классовыми компонентами. Они предоставляют более лаконичный и декларативный способ определения компонентов. На рисунке 3.8 показан реализованный компонентный подход на примере компонента «Modal».

Функциональные компоненты хорошо подходят для простых компонентов без состояния, которые просто отображают данные.

```
import React from 'react'; 6.9k (gziped: 2.7k)

import './modal.scss';

export default function Modal({ isModalOpen, onClose, children, title }) {
  return isModalOpen && (
    <div className="modal">
      <div onClick={onClose} className="overlay"></div>
      <div className="modal-content">
        <h2 className="modal__title">{title}</h2>
        {children}
        <button className="close-modal" onClick={onClose}>
          X
        </button>
      </div>
    </div>
  );
}
```

Рисунок 3.8 – Пример функционального компонента

В данном дипломном проекте для стилизации и внешнего вида компонентов используется расширение языка CSS – SCSS (Sassy CSS).

Это расширение добавляет дополнительные функциональные возможности и синтаксический сахар для создания стилей веб-страниц. SCSS является одним из диалектов препроцессоров CSS, который позволяет использовать переменные, миксины, вложенность правил, операторы и другие возможности, которые упрощают и улучшают разработку стилей.

У каждого компонента его стиль находится в той же папки, для упрощения понимания кода. Далее с помощью webpack все стили компилируются в один общий css файл.

SCSS позволяет импортировать другие SCSS файлы, что упрощает организацию стилей и их повторное использование. Это позволяет разделить стили на логические блоки и импортировать только необходимые файлы.

В папке «styles» дипломного проекта находятся миксины. Миксины в SCSS позволяют определить группу стилей, которые могут быть повторно использованы в разных частях кода. Миксины можно передавать аргументы, что делает их гибкими и настраиваемыми. Примеры миксинов для стилизации шрифтов изображены на рисунке 3.9, а их применение на рисунке 3.10.

```
@mixin typography-primary {  
  font-family: 'Raleway', sans-serif;  
  font-size: 18pt;  
}  
  
@mixin typography-primary-hight {  
  @include typography-primary();  
  font-size: 22pt;  
}  
  
@mixin typography-primary-bold {  
  @include typography-primary();  
  font-weight: bold;  
}
```

Рисунок 3.9 – Пример миксина

```
.header-main {  
  font-weight: 600;  
  
  display: flex;  
  align-items: center;  
  justify-content: end;  
  
  width: 100%;  
  
  @include typography-small;  
  gap: 50px;  
}
```

Рисунок 3.10 – Пример использования миксина

Уровень отображения инкапсулирует весь код, относящийся к пользовательскому интерфейсу и содержит все связанные с ним компоненты. Слой отображения состоит из следующих модулей:

- корневая папка: основная папка проекта, которая содержит все остальные файлы и папки. В ней находятся конфигурационные файлы, такие как `package.json`, `webpack.config.js`, а также файлы запуска или скрипты проекта.
- папка исходного кода (`src`): папка, в которой содержится весь исходный код проекта. Внутри папки исходного кода находятся все файлы и папки, связанные с разработкой приложения;
- папка стилей (`styles`): отдельная папка для хранения стилей приложения и миксинов, располагается в папке исходного кода;
- папка ресурсов (`static`): в этой папке содержатся ресурсы, такие как изображения, шрифты, видео;
- папка хранилища (`store`): в этой папке хранится компонент общего состояния;
- папка базовых компонентов (`components`): в данной папке хранятся компоненты, которые используются в разных модулях приложения, также очень часто компоненты из этой папки могут беспрепятственно повторно использоваться в других проектах;
- папка констант (`constants`): папка, хранящая статические данные приложения такие как маршруты для навигации или запросов на сервер;
- папка страниц (`pages`): в этой папке содержатся компоненты, которые отображают одну конкретную страницу приложения;
- папка сервисов (`helpers`): в данной папке хранятся функции для запросов на сервер, обновления состояния приложения или выполнения другой дополнительной логики;
- папка срезов (`features`): в этой папке хранятся компоненты, разделенные по принадлежности к определенной странице.

Главное преимущество структурированной структуры папок в проекте состоит в том, что она облегчает навигацию, обслуживание и разработку, делая код более организованным и понятным.

Слой отображения в React отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем. Он состоит из компонентов, которые получают данные из слоя бизнес-логики или хранилищ данных и отображают их на экране. Слой отображения организован в виде иерархии компонентов, где каждый компонент представляет отдельную часть пользовательского интерфейса.

Слой отображения содержит состояние компонентов, которое используется для отслеживания и обновления данных, специфичных для компонента, например, при добавлении или удалении карточек.

В данном приложении слой отображения взаимодействует с внешним сервисом, таким как Google Translate API, и библиотекой ``react-reader`` для работы с документами.

Взаимодействие с внешним миром осуществляется через сетевые запросы, таким образом обращение к Google Translate API осуществляется через fetch-запрос, в котором передаются слово или фраза и языки необходимые для корректного перевода.

3.3.5 Webpack

Webpack играет важную роль в React-приложении. Он является сборщиком модулей, который выполняет ряд задач, связанных с разработкой и разворачиванием приложения.

Webpack позволяет разработчикам разбивать код на модули и импортировать их в других файлах. Он обрабатывает зависимости между модулями и создает граф зависимостей, чтобы понять, какие модули должны быть включены в сборку. Затем webpack собирает все модули вместе в единый файл или несколько файлов (в зависимости от конфигурации), которые могут быть загружены и запущены в браузере.

Webpack также может применять различные трансформации к коду, такие как оптимизация, минификация, обработка изображений и другие, чтобы улучшить производительность и размер финальной сборки.

Webpack позволяет разделить код на несколько частей, чтобы улучшить производительность и оптимизировать загрузку приложения. С помощью динамического импорта и разделения кода webpack может создавать отдельные файлы для различных частей приложения. Это позволяет загружать только необходимые части кода по мере необходимости, что уменьшает время загрузки и улучшает производительность приложения.

Webpack может обрабатывать и импортировать различные типы ресурсов, таких как стили CSS, изображения, шрифты и другие. Он может использовать соответствующие загрузчики для обработки и включения ресурсов в сборку, например, преобразовывать их в код, встраивать в CSS или копировать в выходную директорию, пример такого загрузчика продемонстрирован на рисунке 3.11.

В целом, webpack значительно облегчает разработку и разворачивание React-приложений, предоставляя инструменты для сборки, трансформации, оптимизации и управления зависимостями модулей. Он помогает сделать приложение более эффективным, модульным и быстрым.

```
function generateStyleLoader(loaderName, isDevelopmentMode) {
  return [
    MiniCssExtractPlugin.loader,
    {
      loader: 'css-loader',
      options: {
        url: false,
        modules: false,
        importLoaders: 2,
        sourceMap: isDevelopmentMode
      }
    },
    {
      loader: 'postcss-loader',
      options: {
        sourceMap: isDevelopmentMode
      }
    },
    {
      loader: 'resolve-url-loader',
      options: {}
    },
    {
      loader: loaderName,
      options: {
        sourceMap: true,
        sassOptions: {
          includePaths: ['src', 'node_modules']
        }
      }
    }
  ];
}
```

Рисунок 3.11 – Webpack лоадер для обработки стилей

3.4 Серверное API

Серверное API (Application Programming Interface) – интерфейс, предоставляемый сервером, который позволяет взаимодействовать с сервером и обмениваться данными между клиентским приложением и сервером. Серверное API определяет набор эндпоинтов (URL-адресов) и протоколов коммуникации, которые клиентское приложение может использовать для отправки запросов и получения ответов от сервера.

Эндпоинты представляют собой конечные точки API, к которым клиентское приложение может обращаться для выполнения определенных операций. Каждый эндпоинт имеет свой уникальный URL-адрес и может принимать параметры запроса для более точной настройки операций.

Определяют различные подходы и протоколы, которые могут использоваться при разработке и взаимодействии с серверными API. В данном дипломном проекте был выбран RESTful API подход. Он основан на архитектурном стиле REST, который определяет принципы для построения масштабируемых и гибких систем. RESTful API использует стандартные HTTP-методы (GET, POST, PUT, DELETE) для выполнения операций чтения, создания, обновления и удаления данных. Он работает с ресурсами, представленными в виде URL-

адресов, и использует коды состояния HTTP для обозначения результатов операций.

Backend-приложение, написанное на C# с использованием фреймворка ASP.Net, представляет собой API, которое используется React-приложением. API определяет набор команд, которые пользователи вызывают для получения информации.

Без использования API пришлось бы разрабатывать отдельные протоколы связи для каждого стороннего программного обеспечения или службы, с которыми нужно было бы взаимодействовать. Это было бы сложным и трудоемким процессом, а также сделало бы программное обеспечение более уязвимым к изменениям. Создание API позволяет различным системам общаться друг с другом.

Например, если мы в данный момент разрабатываем только веб-приложение, но позже решим создать мобильное приложение для этого проекта, которому также требуется взаимодействие с данными на сервере, то без API нам пришлось бы разрабатывать протокол взаимодействия и код подключения к серверу заново. Но благодаря наличию API это не требуется. Мы просто будем использовать предоставленные методы API - те же самые, что и для веб-сайта.

Для реализации бэкенда используется ASP.Net фреймворк. Благодаря его кроссплатформенности, у нас есть более широкий спектр операционных систем, которые можно использовать для развёртывания.

3.4.1 Конфигурация проекта

Конфигурация проекта на ASP.NET позволяет определить и настроить параметры, используемые приложением. Это включает настройки баз данных, внешних сервисов, режимы работы приложения, ключи API и другие переменные окружения. ASP.NET предоставляет гибкие возможности для управления конфигурацией, чтобы облегчить настройку и изменение параметров приложения без необходимости внесения изменений в сам код.

В ASP.NET используется файл конфигурации `appsettings.json`, который содержит набор ключ-значение пар для настройки приложения. В этом файле определяются собственные настройки, а также используются предопределенные секции, такие как `ConnectionStrings` для строк подключения к базе данных и `Logging` для журналирования процессов приложения.

Также файл конфигурации используется для аутентификации с помощью JWT-токена. Именно в этом файле задаются `AccessTokenSecret`, `RefreshTokenSecret`, `AccessTokenExpirationMinutes` и другие значения.

Доступ к этим настройкам в коде приложения получается с помощью `IConfiguration`, который внедряется в сервисы или контроллеры приложения.

Конфигурация проекта на ASP.NET предоставляет мощный механизм для управления настройками приложения, обеспечивая гибкость и переносимость приложения между различными средами и облегчая внесение изменений в настройки без необходимости перекомпиляции кода.

3.4.2 ASP.NET MVC

ASP.NET MVC (Model-View-Controller) является популярным фреймворком для разработки веб-приложений на платформе ASP.NET.

В случае, когда слой отображения выполняется React-приложением, роль представления в ASP.NET MVC может быть изменена. React-приложение вместо того, чтобы использовать представления Razor или HTML шаблоны для отображения данных, является основным механизмом отображения пользовательского интерфейса.

Контроллеры в ASP.NET MVC остаются ответственными за обработку запросов и взаимодействие с моделью. Они возвращают данные в формате JSON, затем эти данные используются React-приложением для отображения данных и взаимодействия с пользователем.

React-приложение интегрировано в ASP.NET MVC проект с использованием React-компонента. Контроллеры в ASP.NET MVC обслуживают маршруты, которые перенаправляют запросы на соответствующие контроллеры и действия, где обрабатываются запросы и возвращаются данные для React-приложения. Все это задается в файле Program.

3.4.3 Модель

В архитектуре ASP.NET MVC слой Модели отвечает за бизнес-логику и управление данными приложения. Он представляет собой совокупность классов, сервисов и компонентов, которые моделируют объекты и операции предметной области приложения.

Слой Модели содержит классы и компоненты, которые реализуют бизнес-логику приложения. Это могут быть классы, представляющие сущности предметной области, такие как пользователи, продукты или заказы. Они содержат логику валидации данных, обработку бизнес-правил и другие операции, связанные с бизнес-логикой приложения. Также слой Модели обеспечивает доступ и управление данными приложения. Он включает классы и компоненты, отвечающие за доступ к базе данных, взаимодействие с внешними сервисами или API, а также сохранение, обновление и извлечение данных. Это может включать ORM (Object-Relational Mapping) инструменты, такие как Entity Framework, для работы с базой данных.

Еще слой Модели обычно содержит логику валидации данных, чтобы гарантировать правильность и целостность данных, вводимых пользователем. Это может включать проверку формата данных, обязательных полей, уникальности и других правил валидации.

В данном дипломном проекте все модели данных хранятся в папке Entities. Для работы с этими моделями и базой данных были введены два дополнительных класса Repository и Specification.

В целом, слой Модели в ASP.NET MVC играет ключевую роль в разработке бизнес-логики и управлении данными приложения. Он помогает разделить ответственности и создать модульную, тестируемую и поддерживаемую архитектуру приложения.

После определения моделей следует выбрать хранилище данных этих моделей. В проекте будет использоваться PostgreSQL Server, для работы с которым компания Microsoft рекомендует использовать ORM-технология Entity Framework.

3.4.4 Entity Framework

Entity Framework (EF) является объектно-реляционным отображением (ORM) в ASP.NET Core. Он предоставляет удобный способ взаимодействия с базой данных, позволяя разработчикам работать с объектами и сущностями, а не с непосредственным SQL кодом.

Позволяет создавать модели данных с помощью классов, называемых сущностями (entities). Каждая сущность представляет таблицу в базе данных, а свойства класса соответствуют столбцам таблицы. С помощью атрибутов или Fluent API можно задавать дополнительные настройки, такие как ограничения целостности и связи между таблицами.

В процессе работы с Entity Framework необходимо использовать контекст базы данных, то есть класс унаследованный от класса `Microsoft.EntityFrameworkCore.DbContext` для взаимодействия с базой данных. Контекст представляет собой сессию работы с данными и обеспечивает операции, такие как создание, чтение, обновление и удаление записей в базе данных. Он также отслеживает изменения сущностей и автоматически генерирует соответствующие SQL запросы для выполнения операций.

Entity Framework поддерживает создание базы данных и миграции схемы данных. С помощью механизма миграций можно автоматически обновлять структуру базы данных при изменении модели данных, сохраняя существующие данные. Это облегчает управление схемой базы данных в процессе разработки и обновлении приложения.

3.4.5 Контроллеры

Слой контроллера в ASP.NET MVC представляет собой часть архитектурного паттерна MVC (Model-View-Controller) и отвечает за обработку входящих HTTP-запросов, координирует взаимодействие между моделью и представлением и обеспечивает разделение ответственности между компонентами приложения. В шаблоне MVC контроллер является начальной точкой входа в приложение и представляет собой класс на языке C#, который наследуется от абстрактного базового класса `Microsoft.AspNetCore.Mvc.Controller`.

Контроллер принимает входящие HTTP-запросы от клиента и определяет, какое действие должно быть выполнено для обработки запроса. Он анализирует маршрут и метод HTTP, чтобы определить, какое действие следует

вызвать. Также он выполняет роль координатора между моделью и представлением. Контроллер получает данные из модели или передает данные в модель для обновления состояния приложения. Контроллер также определяет, какие данные должны быть переданы представлению для отображения пользователю.

Контроллер содержит логику, необходимую для обработки конкретного запроса. Это может быть проверка данных, взаимодействие с внешними сервисами, выполнение бизнес-правил и другие операции, связанные с обработкой запроса.

Смысл слоя контроллера заключается в обеспечении разделения ответственности и упорядоченного потока обработки запросов в приложении. Контроллеры помогают организовать код приложения, улучшают его модульность и позволяют легко добавлять новые функции и изменять существующую логику без влияния на другие компоненты.

3.4.6 Сервисы

В контексте ASP.NET MVC, сервисы представляют собой классы, которые выполняют определенные функции и бизнес-логику в приложении. Они предоставляют дополнительные возможности для обработки данных и взаимодействия с внешними системами. Сервисы часто используются для выполнения операций, которые не принадлежат прямо модели (Model) или контроллеру (Controller).

Сервисы помогают разделить ответственность и логику приложения. Они выносят специфическую бизнес-логику из контроллера, что делает код более модульным и упрощает его тестирование. Также сервисы могут быть использованы в разных частях приложения и даже в разных контроллерах. Это позволяет повторно использовать код и избегать дублирования функциональности. Сервисы часто используются для взаимодействия с внешними системами, такими как базы данных, API сторонних сервисов, почтовые сервисы и другие. Они инкапсулируют логику взаимодействия с этими системами, обеспечивая чистый интерфейс для контроллеров и моделей. Использование сервисов делает приложение более расширяемым. Если вам потребуется добавить новую функциональность или изменить существующую, вы можете внести изменения только в соответствующие сервисы, не затрагивая другие компоненты приложения.

Примеры типичных сервисов в ASP.NET MVC могут включать сервисы аутентификации и авторизации, сервисы доступа к данным (Data Access Services), сервисы уведомлений, сервисы логирования и т.д. В рамках данного приложения используются следующие сервисы: UserService, CardService, BookService и другие. Сервисы вызывают методы репозитория для взаимодействия с базой данных.

3.4.7 Репозиторий

В ASP.NET репозиторий является шаблоном проектирования, который используется для управления доступом к данным. Он предоставляет абстракцию между слоем модели и слоем доступа к данным, обеспечивая упрощенный доступ к данным и управление операциями CRUD (Create, Read, Update, Delete).

Репозиторий обеспечивает единый интерфейс для доступа к данным, скрывая детали реализации слоя доступа к данным. Он предоставляет методы для выполнения операций чтения, записи, обновления и удаления данных. Также репозиторий помогает разделить ответственность между слоем модели и слоем доступа к данным. Он отделяет логику доступа к данным от бизнес-логики модели, что делает код более модульным и упрощает его тестирование. Как известно, репозиторий предоставляет слой абстракции, который облегчает внесение изменений в слое доступа к данным. Если вы решите изменить источник данных или технологию доступа к данным, вам не придется изменять код везде, где используется репозиторий. Вместо этого вы сможете внести изменения только в реализацию репозитория.

В данном приложении класс репозиторий реализован с помощью универсальных шаблонов TContext и TEntity. Это позволяет в будущем при расширении приложения и добавлении других контекстов беспрепятственно осуществлять сообщение с базой данных. Особенностью TEntity является то, что эта сущность обязательно должна реализовывать интерфейс IHasId.

В целом, репозиторий в ASP.NET является полезным инструментом для управления доступом к данным и обеспечения модульности и гибкости в приложении. Он помогает разделить логику доступа к данным от других компонентов приложения и облегчает тестирование и поддержку изменений.

3.4.8 Спецификация

Спецификации в контексте разработки программного обеспечения обычно относятся к паттерну проектирования, который используется для формулирования бизнес-правил или условий, описывающих характеристики объектов или данных. Спецификация определяет набор правил или критериев, которые должны быть выполнены для удовлетворения определенного условия. Спецификации часто используются в слое модели для проверки данных, фильтрации или выборки объектов в соответствии с определенными критериями. Спецификации могут быть использованы для определения условий поиска, фильтрации коллекций объектов или определения бизнес-правил в контексте доменной модели.

Основная идея паттерна «спецификация» заключается в том, чтобы отделить логику проверки условий от самих объектов, на которые эти условия применяются. Вместо включения логики проверки внутрь объектов, спецификация представляет собой отдельный объект, реализующий набор правил или

критериев. Это позволяет создавать композиции спецификаций для создания более сложных условий и динамического формирования запросов.

Преимущества использования паттерна «спецификация» включают:

- увеличение гибкости и переиспользования: спецификации могут быть легко комбинированы и повторно использованы для создания различных условий и запросов;
- улучшенная читаемость и понимание кода: бизнес-правила выражены в виде отдельных объектов, что делает код более понятным и поддерживаемым;
- упрощение тестирования: спецификации могут быть легко протестированы независимо от других компонентов системы.

Однако, следует отметить, что применение паттерна «спецификация» может добавить сложность в систему, особенно при работе с большим количеством правил и условий. Поэтому необходимо тщательно оценивать, когда и как использовать этот паттерн, чтобы избежать излишней сложности и ухудшения производительности.

3.4.9 Миграции

Миграции в Entity Framework (EF) – механизм, который позволяет управлять изменениями схемы базы данных и автоматически применять эти изменения при обновлении приложения. Связь EF с миграцией изображен на рисунке 3.11.

Основные концепции и компоненты миграций в Entity Framework:

- модель данных: модель данных представляет структуру базы данных и сущности, с которыми работает приложение. Модель может быть представлена с помощью классов Entity Framework или с помощью подхода Code First, где модель строится на основе классов с атрибутами или конфигурациями;
- миграционные файлы: миграционные файлы представляют собой скрипты или классы, которые описывают изменения в схеме базы данных. В каждом миграционном файле указывается, какие операции должны быть выполнены при обновлении базы данных: создание новых таблиц, добавление или удаление столбцов, изменение ограничений и т. д.;
- контекст данных представляет собой класс, который настраивает доступ к базе данных и определяет набор сущностей, с которыми работает приложение. Контекст данных также отслеживает и применяет миграции при обновлении базы данных;
- команды миграций предоставляются Entity Framework для выполнения операций миграций. С помощью этих команд можно создавать новые миграции, применять миграции к базе данных, откатывать миграции и многое другое.



Рисунок 3.12 – Миграции в Entity Framework

Процесс работы с миграциями в Entity Framework обычно включает следующие шаги:

- Создание и настройка модели данных.
- Создание первоначальной миграции, которая представляет собой текущее состояние базы данных.
- Применение миграций к базе данных для создания или обновления схемы.
- Повторение процесса при необходимости внесения изменений в схему базы данных.

Преимущества использования миграций в Entity Framework:

- миграции позволяют удобно управлять изменениями в схеме базы данных и автоматически применять эти изменения при обновлении приложения;
- миграции позволяют сохранить целостность данных при обновлении схемы базы данных, так как изменения могут быть применены поэтапно и с контролем версий;
- миграции позволяют легко совместно работать над изменениями в схеме базы данных, так как каждая миграция представляет собой отдельный файл или класс, который можно добавлять в систему контроля версий;
- миграции позволяют легко переносить базу данных между различными средами (например, разработка, тестирование, производство), так как изменения могут быть применены автоматически на каждой среде;
- миграции в Entity Framework являются мощным инструментом для управления изменениями в базе данных и обеспечения согласованности схемы данных в приложении.

3.5 База данных

В качестве базы данных данного дипломного проекта была выбрана PostgreSQL, для развёртывания которой проект использует Azure Cloud Storage. Azure Cloud Storage – облачное хранилище данных, предоставляемое Microsoft Azure. Оно предоставляет масштабируемое и надежное хранилище для различных типов данных, которые могут быть использованы в приложениях и сервисах.

Azure предоставляет множество инструментов разработки и управления для разработчиков ASP.NET. Например, Visual Studio и Visual Studio Code

имеют интеграцию с Azure, позволяя вам легко создавать, отлаживать и разворачивать приложения ASP.NET в облаке.

На рисунке 3.13 представлена схема базы данных, на которой изображены основные модели, их свойства и связи. Подробное описание основных таблиц базы данных показано в таблицах.



Рисунок 3.13 – Схема базы данных

Таким образом в 3 главе осуществилось подробное описание процесса проектирования разрабатываемого программного продукта, его архитектуры, компонентов и взаимодействия между ними. Были приведены основные концепции и решения, принятые при проектировании, а также обоснован выбор использованных технологий.

Также в этом разделе была описана общая архитектура программного средства. Были подробно описаны основные модули и компоненты программного средства. Каждый модуль рассматривался в отдельности, описывалась его функциональность, интерфейсы и взаимодействие с другими компонентами системы.

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

4.1 Элементы пользовательского интерфейса

Необходимой деталью приложения, обеспечивающей взаимодействие пользователя с онлайн-платформой, является пользовательский интерфейс. Пользовательский интерфейс – совокупность элементов и средств, которые позволяют пользователю взаимодействовать с программным или аппаратным обеспечением. Он представляет собой точку взаимодействия между пользователем и системой, обеспечивая способ коммуникации, контроля и управления.

Цель пользовательского интерфейса - сделать взаимодействие пользователя с системой интуитивно понятным, удобным и эффективным. Хороший пользовательский интерфейс облегчает выполнение задач пользователя, предоставляет информацию ясно и наглядно, и создает позитивный пользовательский опыт.

React позволяет создавать компоненты, которые могут быть использованы для отображения определенной части пользовательского интерфейса. React использует виртуальный DOM (Document Object Model) для эффективного обновления только измененных частей пользовательского интерфейса. Вместо обновления всего дерева DOM, React оптимизирует процесс обновления, что приводит к более быстрой и эффективной работе интерфейса.

Компоненты группируются в соответствии с секцией, в которой они применяются (user-page, book-page, common и др.).

Рассмотрим компонент под названием `BasePage`, предоставленный в листинге 4.1, который представляет собой базовый компонент для последующего получения кастомизированных под определенные требования и запросы компонентов страниц, таких как `AccountPage`, `LoginPage` и др.

```
export default function BasePage({ children, isLoginPage }) {
  const dispatch = useDispatch();
  const isAuthenticated = JSON.parse(localStorage.getItem('isAuthenticated'));
  const userName = useSelector(state => state.users.username);
  const notifications = useSelector(state => state.users.notifications);

  useEffect(() => {
    (
      async () => {
        if (isAuthenticated) {
          const user = await requestHelper.get(serviceUrls.getUser, {
            method: 'GET',
            headers: {
              'Content-Type': 'application/json',
            },
            credentials: 'include'
          });
        }
      }
    );
  }, [isAuthenticated]);
}
```

```

    }, isAuth, true);

    if (user) {
      dispatch({
        type: 'users/setUserData',
        payload: {
          isAuth: true,
          username: user.firstName
        }
      });
    }
  }

  if (!notifications) {
    const notificationData = await requestHelper.get(
      serviceUrls.getNotifications, {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
        },
        credentials: 'include'
      }, isAuth, true);

    if (notificationData) {
      dispatch({
        type: 'users/setUserNotifications',
        payload: {
          notifications: notificationData
        }
      })
    }
  }
}

})();

if (!isAuth && !isLoginPage) {
  window.location.replace(routes.LOG_IN);
}
}, [isAuth]);

return (
  <div className='base-page'>
    <Header name={userName} />
    <div className='base-page__content'>
      {children}
    </div>
  </div>
);

```

```

        </div>
        <Footer />
    </div>
)
}

```

Листинг 4.1 – Компонент «BasePage»

В листинге выше мы видим компонент, который в качестве параметров принимает «children» и «isLoginPage». В React компонентах, параметр «children» используется для передачи дочерних элементов (или компонентов) внутрь другого компонента. Он позволяет вложенным компонентам получить доступ к содержимому, переданному между открывающим и закрывающим тегами компонента-контейнера. В данном случае «BasePage» выступает в роли компонента-контейнера. В итоге, использование «children» позволяет создавать более гибкие и составные компоненты, которые могут содержать и манипулировать своими дочерними элементами. Параметр «isLoginPage» используется для логического определения, является ли вложенный компонент страницей логирования.

Далее в компоненте используются два хука «useEffect» и «useDispatch». Хуки – новое API, представленное в React версии 16.8, которое позволяет использовать состояние и другие функциональные возможности React в функциональных компонентах.

Хуки позволяют функциональным компонентам иметь свое состояние и использовать другие функциональные возможности, ранее доступные только в классовых компонентах. Хуки также облегчают повторное использование состояния и логики между компонентами.

«useEffect» – хук в библиотеке React, который позволяет выполнять побочные эффекты в функциональных компонентах. Побочные эффекты могут быть такими действиями, как отправка сетевых запросов, обработка подписки на события, выполнение таймеров и другие операции, которые влияют на состояние компонента или взаимодействуют с внешними ресурсами.

Хук «useEffect()» позволяет указать функцию-эффект, которая будет выполнена после каждого рендера компонента. Функция-эффект может возвращать функцию очистки, которая будет вызвана перед удалением компонента или перед выполнением следующего эффекта. Это позволяет контролировать выполнение и очистку побочных эффектов.

Внутри данного хука в компоненте «BasePage» происходит проверка пользователя на аутентификацию и, в случае положительного результата, отправляется «GET» запрос для получения данных пользователя, и если он еще не установлен, то с помощью «dispatch» метода устанавливаются полученные из запроса данные.

«useDispatch» – функция, предоставляемая библиотекой Redux, которая позволяет компонентам React взаимодействовать с хранилищем данных и диспетчеризовать действия (actions).

Хранилище данных в Redux содержит состояние приложения, которое может быть изменено с помощью действий. Действия представляют собой простые объекты, описывающие, что произошло в приложении. Они передаются в функцию «dispatch()» для изменения состояния в хранилище.

Однако для того, чтобы компонент мог отправлять действия, ему необходим доступ к функции «dispatch()». Вот где на помощь приходит «useDispatch()». Эта функция позволяет компонентам получить доступ к экземпляру функции «dispatch()», которую они могут использовать для отправки действий в хранилище.

В приведенном выше листинге кода 4.1 мы импортируем функцию «useDispatch()» из библиотеки «react-redux». Затем мы вызываем «useDispatch()» и сохраняем результат в переменной «dispatch». Внутри компонента мы можем использовать «dispatch» для изменения состояния хранилища.

Сам компонент BasePage состоит из трех основных блоков:

- компонент «Header», изображенный на рисунке 4.1, обычно располагается в верхней части страницы и содержит элементы, такие как навигационное меню, заголовок страницы и другие элементы, которые могут быть важными для навигации и представления контента на сайте;
- компонент «Footer» обычно располагается в нижней части страницы и содержит информацию, которая повторяется на всех страницах сайта, такую как копирайт и другую дополнительную информацию;
- основной блок компонента, в который передается параметр «children», описанный выше.



Рисунок 4.1 – Компонент «Header»

4.2 Регистрация и авторизация пользователя

При регистрации пользователь указывает свои данные такие как адрес электронной почты и пароль, эти данные затем используются при авторизации. Регистрация пользователя происходит в компоненте «SignUpPage». Для регистрации кроме указанных выше данных пользователю нужно указать имя,

фамилию, дату рождения и по желанию выбрать фото для аккаунта. Также для проверки пароля имеется второе поле для подтверждения введенного пароля.

Затем на стороне сервера данные о пользователе принимает «Register» эндпоинт, находящийся в «AuthController». Этот контроллер отвечает за все методы и действия приложения, связанные с регистрацией и авторизацией пользователей.

В данном дипломном проекте для аутентификации и авторизации пользователей используется JWT-токен (JSON Web Token).

JWT-токены позволяют подтвердить подлинность пользователя. При успешной аутентификации пользователь получает JWT-токен, который затем используется для идентификации пользователя при последующих запросах к серверу. Токен содержит информацию, такую как идентификатор пользователя или другие учетные данные. Он состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи (signature).

Использование JWT-токенов позволяет создавать безсостоятельные приложения, которые не требуют хранения состояния на сервере. Клиент и сервер могут обмениваться токенами для выполнения аутентификации и авторизации, что делает JWT-токены популярным инструментом для обеспечения безопасности в веб-приложениях.

В ASP.NET для создания и проверки JWT токенов используется библиотека System.IdentityModel.Tokens.Jwt.

При получении JWT токена от клиента (из заголовка Authorization), в ASP.NET происходит его проверка на подлинность и целостность.

Когда пользователь заходит в приложение, происходит проверка его на существование в системе, если пользователь уже зарегистрирован, то создается объект класса «AuthenticatedInfo», который хранит в себе «AccessToken» и «RefreshToken». Эти токены возвращаются на сторону клиента и сохраняются в localStorage для дальнейшего обмена данных между клиентом и сервером.

```
localStorage.setItem('access', JSON.stringify(response.accessToken))
localStorage.setItem('refresh', JSON.stringify(response.refreshToken))
```

Листинг 4.2 – Сохранение токенов на стороне клиента

У зарегистрированного и авторизованного пользователя появляется возможность управления настройками своего аккаунта, где он может указать свой родной язык, выбрать нужны ли ему уведомления или изменить имя с фамилией.

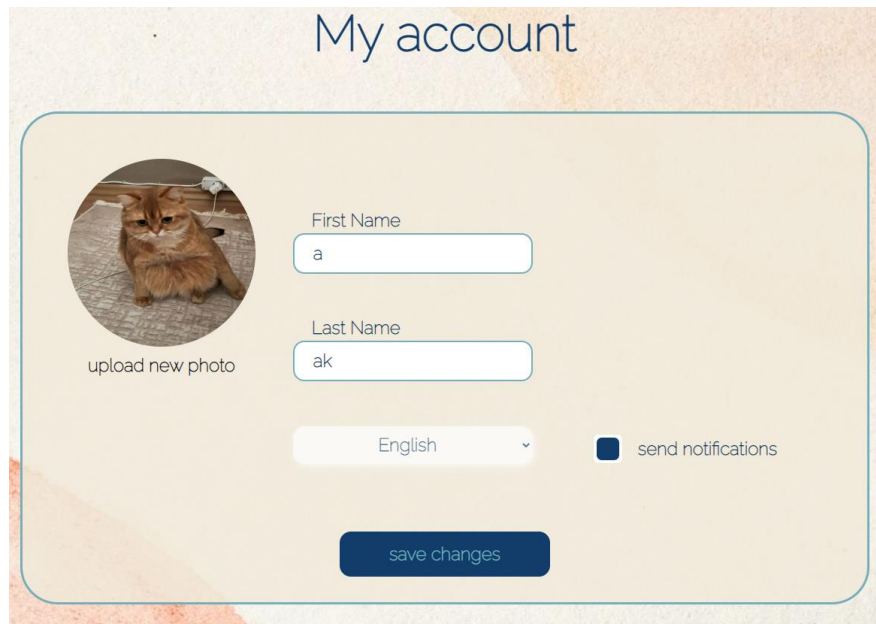


Рисунок 4.2 – Форма настройки аккаунта

4.3 Работа с текстовыми документами

Отличительной чертой данного приложения является возможность самому добавлять текстовые документы для чтения и перевода.

Для этого пользователь должен перейти на страницу его загрузок и нажать на кнопку «add». Затем выбрать книгу, которую хочет добавить. После этого на экране появляется модальное окно, где пользователь может указать жанр, язык, название, автора и краткое описание книги. Выбор языка является важным шагом при добавлении книги, так как на основе языка книги в дальнейшем будет делаться перевод карточек.

После подтверждения формы создания книги, книга отправляется на сервер, где в предоставленном в листинге 4.3 контроллере происходит обработка полученных из запроса данных.

```
public async Task<Book> CreateBook(BookData bookData, int userId)
{
    var book = new Book
    {
        Title = bookData.Title,
        Description = bookData.Description,
        Author = bookData.Author,
        UserId = userId,
        GenreId = bookData.Genre,
        LanguageId = bookData.Language,
        Content = fileHelper.GetFilesBytes(bookData.Book)
    };
}
```

```

if (bookData.Cover != null )
{
    book.Cover = fileHelper.GetFilesBytes(bookData.Cover);
}

return await bookRepository.Create(book);
}

```

Листинг 4.3 – Создание книги на стороне сервера

Для дальнейшего чтения книги, нужно перейти на страницу книги и нажать на кнопку «read». После нажатия пользователь оказывается на странице с книгой, где ему доступны следующие возможности:

- чтение книги;
- переход на следующую страницу нажатием на кнопку;
- открытие содержания книги, изображено на рисунке 4.3;
- переход на определенную главу с помощью содержания книги;
- создание карточек с переводом.

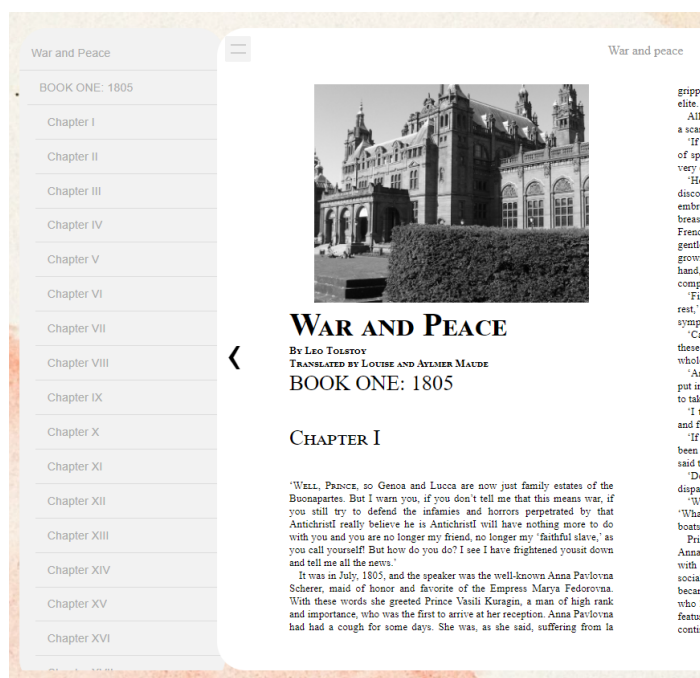


Рисунок 4.3 – Страница чтения книги

Таким образом у авторизованного пользователя появляется возможность добавлять книги для того чтобы в будущем их читать и создавать по ним карточки с переводами в случае необходимости.

4.4 Работа с флеш-карточками

Рассмотрим, каким образом создаются карточки. При чтении пользователь выбирает неизвестное ему слово. После того, как слово было выделено, как показано на рисунке 4.4, срабатывает «useEffect» хук, внутри которого формируется запрос для отправки в «Google Translate API», приведенный в листинге 4.4. Язык, с которого нужно переводить устанавливается из данных о книге. А язык, на который нужно осуществить перевод, устанавливается из персональных настроек пользователя.

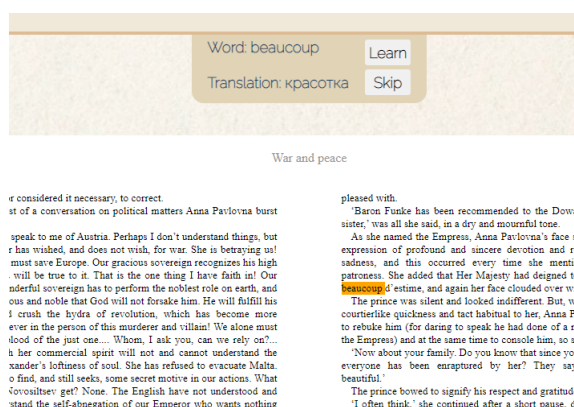


Рисунок 4.4 – Процесс выделения слова для перевода

Также приложение предоставляет перевод не только одного слова, но и целых фраз.

```
const currentUrl = Object.assign(url).toString()
.concat(`&q=${encodeURIComponent(
generateRequestUrl(selections.text))}&source=${lang}&target=${toLang}`);

const translate = await fetch(currentUrl, {
method: 'GET',
headers: {
"Content-Type": "application/json",
Accept: "application/json"
}
});

const { data } = await translate.json();
const { translatedText } = await data.translations[1];
```

Листинг 4.4 – Отправка запроса на перевод

После перевода пользователь может решить, добавлять ему слово для дальнейшего изучения или пропустить его. Подробный алгоритм представлен на рисунке 4.5.

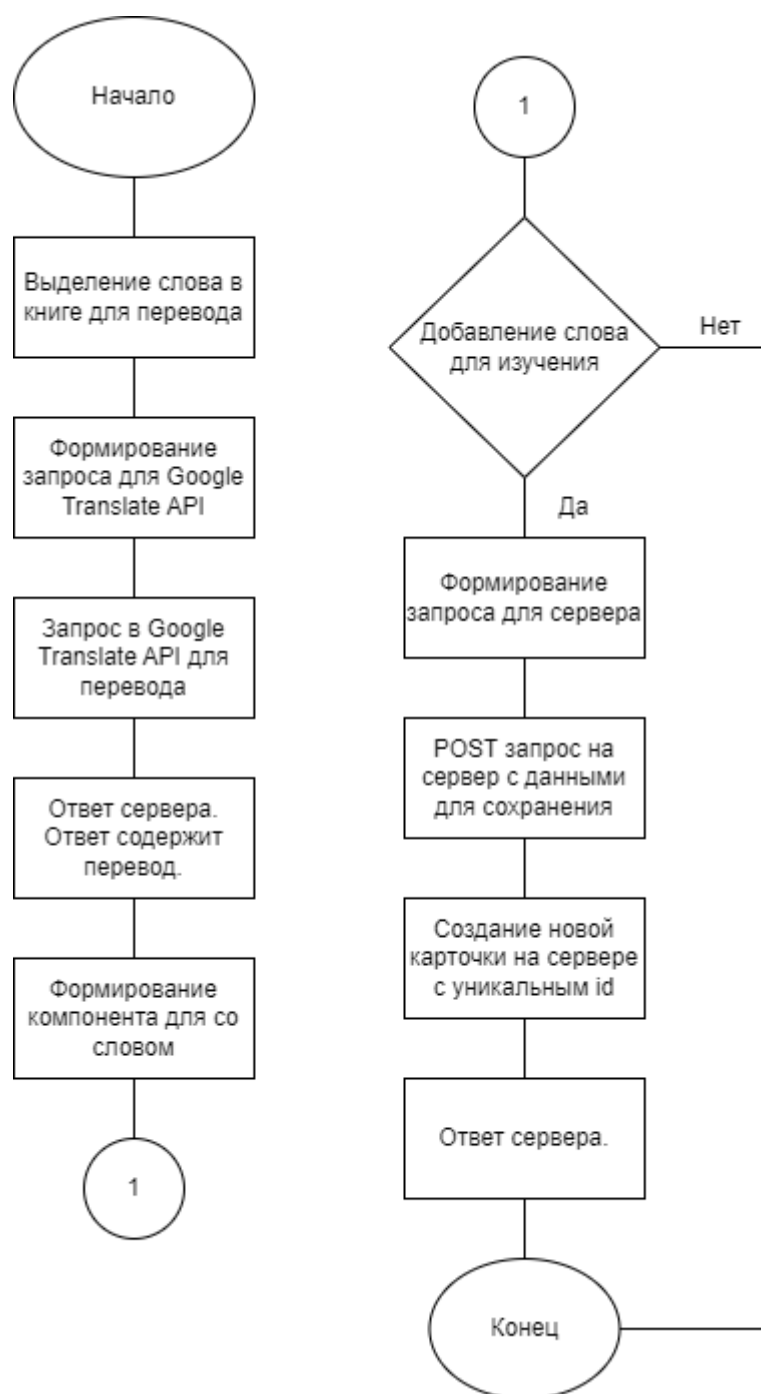


Рисунок 4.5 – Алгоритм создания флеш-карточки

Для того чтобы повторить добавленные слова в виде карточек, пользователь должен перейти на страницу карточек. На эту страницу можно попасть двумя способами: через ссылку на страницу в главном пользовательском меню или через иконку с уведомлениями в хэдре приложения. Однако, попадая на

страницу карточек по второму пути, пользователь окажется на странице только с теми карточками, которые настало время повторить.

Во время повторения пользователь может отметить карточку как повторенную или как выученную, как изображено на рисунке 4.7. Процесс обновления карточки отражен на рисунке 4.6.

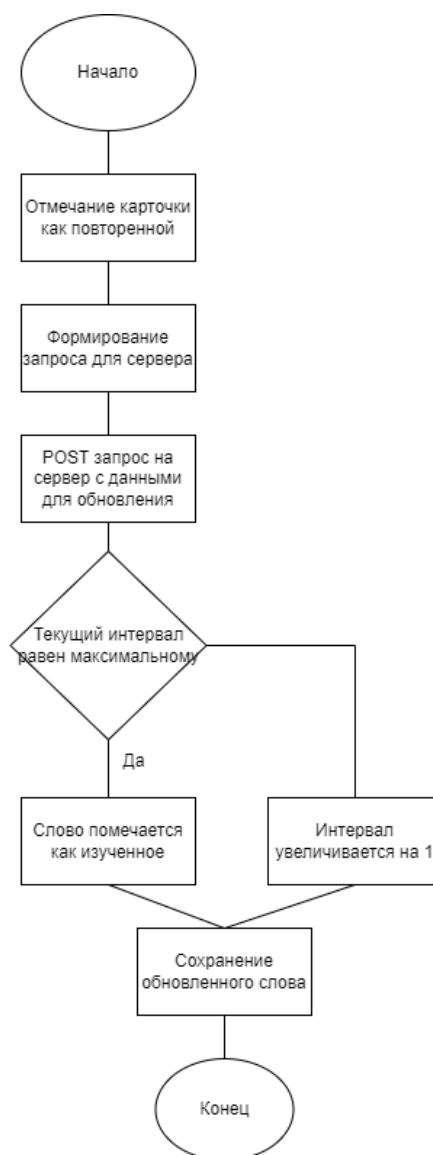


Рисунок 4.6 – Алгоритм обновления флеш-карточки

Также важным моментом создания карточек является создание интервала, через который карточка доступна к повторению. За основу этой идеи был взят метод Лейтнера, суть которого заключается в повторении изученного материала через разные промежутки времени.

Система Лейтнера – методика эффективного запоминания и повторения информации, основанная на принципах активного повторения и распределе-

ния повторений во времени. Она была разработана немецким психологом Себастьяном Лейтнером в середине XX века и широко применяется в обучении и изучении различных предметов и языков.

Основная идея системы Лейтнера заключается в том, чтобы эффективно распределять повторения материала во времени, фокусируясь на сложных и недавно изученных понятиях. Система основывается на использовании колоды карточек.

В ходе разработки данного дипломного проекта было решено взять за начальный интервал для повторения 15 минут. Это значит, что через 15 минут после добавления, пользователь сможет увидеть добавленную карточку, среди карточек для повторения. После того, как пользователь отметит карточку как повторенную, на сервере отработает логика, которая заменит интервал для повторения карточки на более долгий, и следующий раз пользователь сможет увидеть эту карточку только по истечении поставленного интервала.

Если пользователь попадает на страницу карточек через главное меню пользователя, то он получает список всех карточек, которые он когда-либо создавал. Здесь пользователь может фильтровать карточки по языку или по книге, из которой они были добавлены.

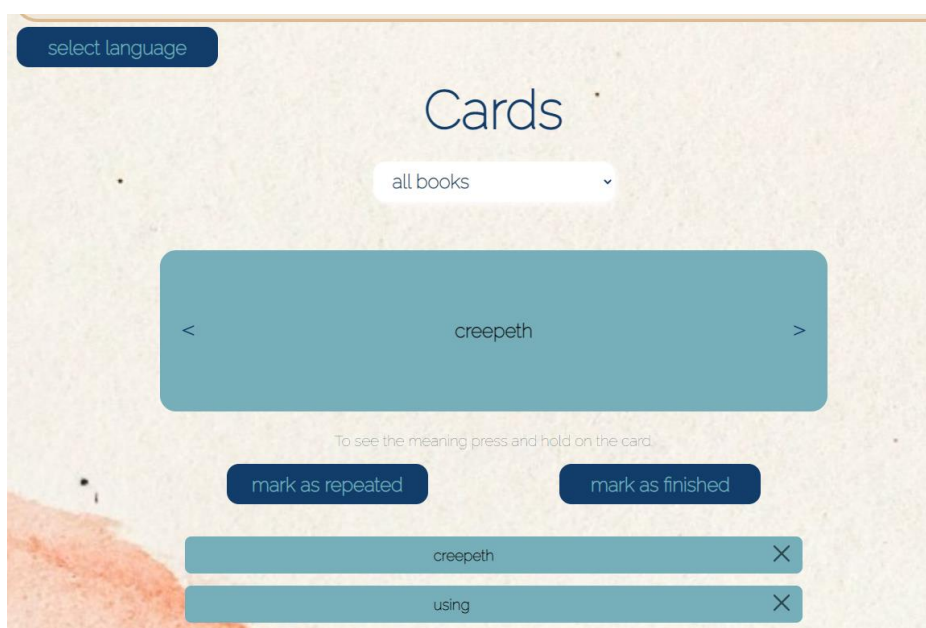


Рисунок 4.7 – Страница с карточками для повторения

Таким образом, в рамках данной главы было реализовано приложение, являющееся объектом разработки данного дипломного проекта. В процессе разработки были рассмотрены особенности реализации систем, таких как React-приложение и сервер на ASP.NET, а также их взаимодействие. В результате описанной в данной главе разработки, было создано рабочее программное средство, готовое для дальнейшего тестирования.

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Данная глава является одной из ключевых составляющих процесса разработки и обеспечения качества программного продукта. Тестирование играет важную роль в выявлении ошибок, проверке функциональности, надежности и производительности приложения, а также гарантирует соответствие программного средства требованиям и ожиданиям пользователей.

Поскольку онлайн-платформа получает данные, взаимодействуя с Google Translate API, а потом использует эти данные, имеет смысл протестировать эту часть приложения.

Также приложение сохраняет и получает данные при помощи сервера, написанного на ASP.NET, поэтому также будет иметь смысл протестировать бэкенд часть дипломного проекта. Тестирование данной части будет производиться с использованием инструмента для тестирования API – Postman.

5.1 Тестирование корректности создания карточки

Процесс создания карточки был описан выше, но следует рассмотреть достоверность перевода выделенных фрагментов. Для проверки этого следует взять переводы с разных языков, получаемые с помощью Google Translate API, и сравнить их с переводами, получаемыми с помощью другого сервиса переводов – Яндекс.Переводчик. В качестве тестового экземпляра было выбрано слово из книги на французском языке. На рисунке 5.1 и 5.2 видим, что лексический перевод слов совпадает.

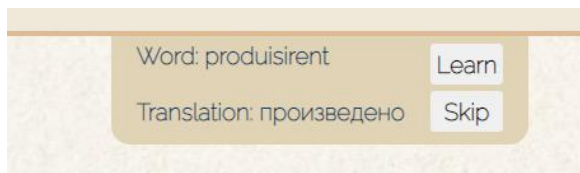


Рисунок 5.1 – Перевод слова с французского языка

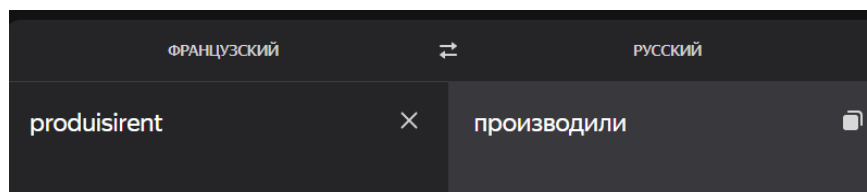


Рисунок 5.2 – Перевод слова в Яндекс.Переводчик

Далее рассмотрим перевод фразы. Для этого возьмем книгу на английском языке и выделим осмысленную фразу.

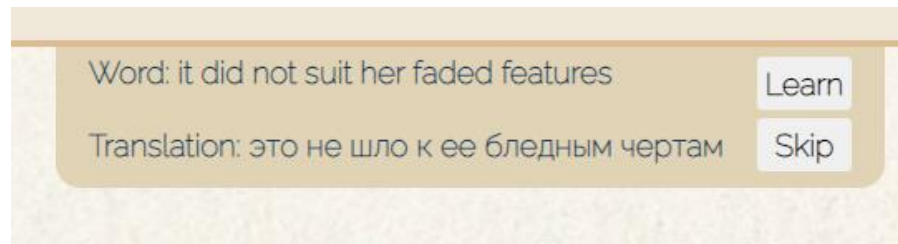


Рисунок 5.3 – Перевод фразы

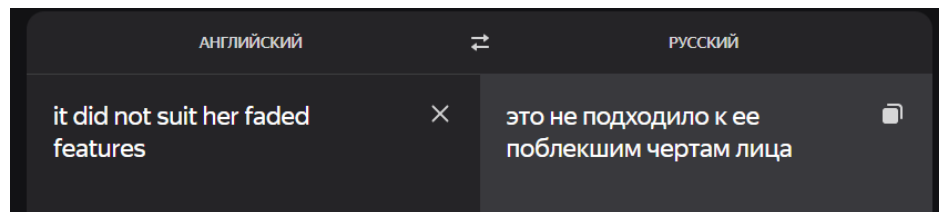


Рисунок 5.4 – Перевод фразы в Яндекс.Переводчик

Как видим в предоставленных рисунках 5.3 и 5.4 серверное API прекрасно справляется с поставленными задачами.

5.2 Тестирование создания пользователя

Для того чтобы протестировать как создается юзер нужно отправить POST запрос с данными приведенными в листинге 5.1.

```
{
  "FirstName": "Anastasia",
  "LastName": "Kruhlaya",
  "Email": "anastasia@gmail.com",
  "BirthDate": "2002-02-27",
  "Image": null,
  "Password": "secret_password"
}
```

Листинг 5.1 – Данные для создания пользователя

Ожидаемый результат: статус Created и создание пользователя в базе данных.

Фактический результат: статус Created и пользователь, изображенные на рисунке 5.5, и добавление пользователя в базу данных, изображенное на рисунке 5.6

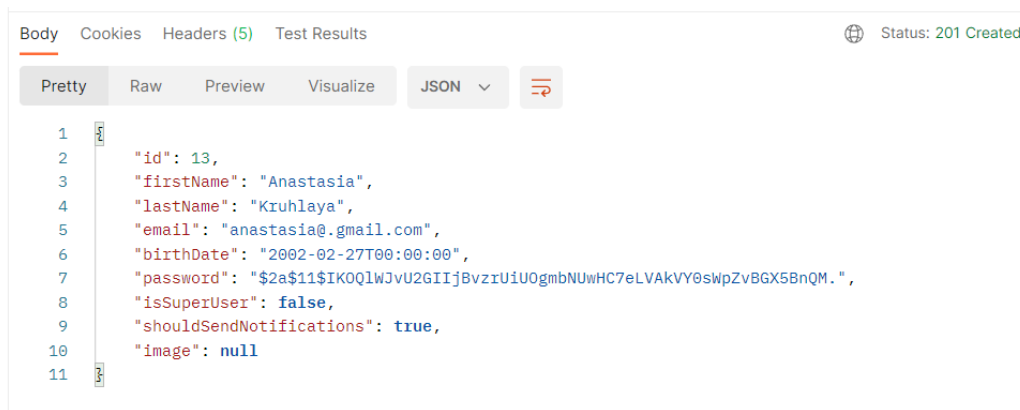


Рисунок 5.5 – Ответ с сервера

12	a@gmail.com	password	a@gmail.com
13	Anastasia	Kruhlaya	anastasia@gmail.com

Рисунок 5.6 – Созданный в базе данных пользователь

Таким образом видим, что пользователь корректно создается.

Рассмотрим процесс авторизации пользователя. Для этого рассмотрим POST запрос на эндпоинт «/auth/login» со следующими данными:

```

{
  "Email": "anastasia@gmail.com",
  "Password": "secret_password"
}

```

Листинг 5.2 – Данные для авторизации

Ожидаемый результат: статус Ок и токены для дальнейшего взаимодействия между клиентом и сервером.

Фактический результат: статус Ок и токены, изображенные на рисунке 5.7.

для создания книги. В тестовом случае назовем книгу «War and Piece» и укажем жанр «Classics».

Ожидаемый результат: книга добавится с указанными данными и отобразится в списке загруженных книг.

Фактический результат: книга добавилась с указанными данными, изображенными на рисунке 5.9, и отображается в списке добавленных книг, что можно увидеть на рисунке 5.10.



Рисунок 5.9 – Информация о книге

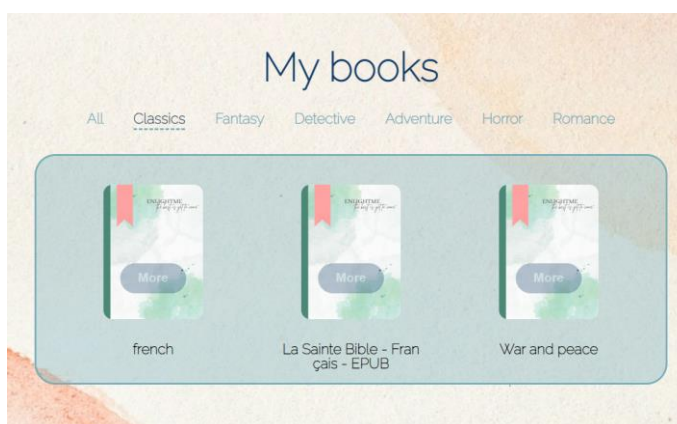


Рисунок 5.10 – Список добавленных книг

Таким образом в данной главе было проведено тестирование спроектированного и реализованного дипломного проекта.

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ОНЛАЙН- ПЛАТФОРМЫ ДЛЯ ИЗУЧЕНИЯ ИНОСТРАННЫХ ЯЗЫКОВ

6.1 Характеристика онлайн-платформы, разрабатываемой для реализации на рынке

Разрабатываемая в дипломном проекте онлайн-платформа для изучения иностранных языков предназначена для чтения книг на иностранном языке и создания карточек, с целью дальнейшего изучения и запоминания новых слов. Онлайн-платформа предназначена для использования широким кругом потребителей.

Основной задачей данного дипломного проекта является разработка актуального и удобного в использовании приложения для чтения текстов на иностранных языках с возможностью создавать карточки с неизвестными словами и их переводами для дальнейшего заучивания новых слов.

Обучение иностранному языку путём чтения книг с переводом – один из наиболее доступных методов для самостоятельного изучения иностранного языка. Таким образом у клиентов появляется возможность самостоятельного изучения иностранного языка в любое удобное время.

Разрабатываемый продукт, предназначенный для решения вышеперечисленных задач, имеет следующие основные функции:

- регистрация пользователя и авторизация в приложении;
- загрузка собственных текстовым материалов;
- создание карточек для неизвестных слов;
- автоматический перевод выделенного текста на иностранном языке;
- сохранение переведённого текста в виде карточки;
- настройка аккаунта пользователя.

Чтение книг и изучение иностранного языка несут множество неоспоримых преимуществ в жизни. Таких как новые знания, удобства в понимании и общении с людьми, не знающими ваш родной язык, улучшение памяти и уменьшение теоретически возможного снижения интеллектуальных способностей.

Таким образом, разрабатываемое приложение становится полезным инструментом, с помощью которого можно будет максимально эффективно заниматься самообразованием.

Продукт разрабатывается для реализации на массовом рынке. Наиболее выгодной для монетизации онлайн-платформы будет модель подписки. У клиентов будет возможность приостановить подписку по своему усмотрению, что предоставляет удобство и простоту в использовании.

6.2 Расчет инвестиций в разработку онлайн-платформы для её реализации на рынке

6.2.1 Затраты на основную заработную плату разработчиков

Расчёт затрат на основную заработную плату команды, занимающейся его разработкой и поддержкой, определяется по следующей формуле:

$$З_0 = K_{\text{пр}} \cdot \sum_{i=1}^n З_{\text{чи}} \cdot t_i \quad (6.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$K_{\text{пр}}$ – коэффициент премий (составляет 1,5);

$З_{\text{чи}}$ – часовая заработная плата i -го исполнителя (руб.);

t_i – трудоемкость работ, выполняемых i -м исполнителем (ч).

Расчётная норма рабочего времени составляет 168 часов. Срок разработки данного приложения – четыре месяца (696 рабочих часов).

Для разработки данного проекта необходимо привлечь следующих специалистов:

- Проектный менеджер (привлекается на начальных этапах разработки проекта, часовой оклад – 21,73 рубля).

- Бизнес-аналитик (привлекается периодически по мере разработки проекта, 15,71).

- Инженер-программист, Frontend React-разработчик (привлекается на половину периода разработки проекта, часовой оклад – 14,21 рубля).

- Инженер-программист, Backend .NET-разработчик (привлекается на весь период разработки проекта, часовой оклад – 15,88 рублей).

- Тестировщик (привлекается на последнюю треть периода разработки, часовой оклад – 13,11 рублей).

Размеры должностных окладов специалистов указаны по состоянию на 31.03.2023.

Расчёт затрат на основную заработную плату команды разработчиков с учётом их трудозатрат по проекту представлены в таблице 6.1.

Таблица 6.1 – Расчет затрат на основную заработную плату команды разработчиков.

Участник команды	Месячный оклад, р.	Часовой оклад, р.	Трудоёмкость работ, ч.	Итого, р.
Проектный менеджер	3 650, 64	21,73	32	695,36
Бизнес-аналитик	2 639, 28	15,71	48	754,08
Frontend React-разработчик	2 387, 28	14,21	168	2 287,28
Backend .NET-разработчик	2 667, 84	15,88	336	5 335,68
Тестировщик	Тестирование ПО	13,11	112	1 468,32
Премия(50%)				5 270,36
Итого затраты на основную заработную плату разработчиков				15 811,08

6.2.2 Затраты на дополнительную заработную плату разработчиков

Затраты на дополнительную заработную плату команды разработчиков определяются по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100} \quad (6.2)$$

где $З_о$ – затраты на основную заработную плату, (р.);

$Н_д$ – норматив дополнительной заработной платы (15%).

Рассчитаем затраты на дополнительную заработную плату:

$$З_д = \frac{15811,08 \cdot 15\%}{100} = 2\,371,66 \text{ р.}$$

6.2.3 Отчисления на социальные нужды

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами Республики Беларусь по формуле:

$$P_{\text{соц}} = \frac{(З_о + З_д) \cdot Н_{\text{соц}}}{100} \quad (6.3)$$

где $Н_{\text{соц}}$ – норматив отчислений на социальные нужды, %, $Н_{\text{соц}} = 34,6\%$.

Сумма отчислений на социальные нужды, согласно формуле (6.3), составит:

$$P_{\text{соц}} = \frac{(15811,08 + 2371,662) \cdot 34,6\%}{100} = 6291,23 \text{ р.}$$

6.2.4 Прочие затраты

Расходы по данной статье включают определяются по формуле:

$$P_{\text{пр}} = \frac{З_о \cdot Н_{\text{пр}}}{100}, \quad (6.4)$$

где $Н_{\text{пр}}$ – норматив прочих затрат в целом по организации.

Приняв значение $Н_{\text{пр}}$ равным 13% и подставив в формулу (6.4) оставшиеся значения, произведем расчет прочих затрат:

$$P_{\text{пр}} = \frac{15811,08 \cdot 13\%}{100} = 2\,055,44 \text{ р.}$$

6.2.5 Расходы на реализацию

Под расходами на реализацию понимают выраженные в денежной форме затраты материальных, трудовых и других видов ресурсов торговых организаций по доведению товаров до конечного потребителя.

Расходы по данной статье рассчитываются по следующей формуле:

$$P_{\text{р}} = \frac{З_о \cdot Н_{\text{р}}}{100}, \quad (6.5)$$

где $Н_{\text{р}}$ – норматив расходов на реализацию, %.

Приняв значение $Н_{\text{р}}$ равным 3% и подставив в формулу (6.5) оставшиеся значения, произведем расчет расходов на реализацию:

$$P_p = \frac{15811,08 * 3\%}{100} = 474,33 \text{ р.}$$

6.2.6 Общая сумма затрат

Общая сумма затрат на разработку онлайн-платформы для изучения иностранных языков находится путем суммирования всех рассчитанных статей затрат. Расчет общей суммы затрат представлен в таблице 6.2.

Таблица 6.2 — Общая сумма затрат на разработку программного обеспечения.

Статья затрат	Сумма, р.
Основная заработная плата команды разработчиков	15 811, 08
Дополнительная заработная плата команды разработчиков	2 371, 66
Отчисления на социальные нужды	6 291,23
Прочие расходы	2 055,44
Расходы на реализацию	474,33
Общая сумма затрат на разработку	27 003,74

Полная себестоимость разрабатываемой онлайн-платформы для изучения иностранных языков составит 27 003,74 рублей.

6.3 Расчет экономического эффекта от реализации онлайн-платформы для изучения иностранных языков на рынке

Экономический эффект для организации-разработчика онлайн-платформы представляет собой прирост чистой прибыли от его продажи на рынке потребителям, величина которого зависит от объема продаж, цены реализации и затрат на разработку программного средства.

Целевой аудиторией разрабатываемого программного средства являются студенты и люди, занимающиеся самообразованием. Так как главным способом монетизации ПО выбрана модель подписки, то после анализа рынка приложений со схожим рабочим функционалом была определена цена для подписки 2,99 р.

После анализа трафика приложений конкурентов, где среднее количество активных пользователей составляет 100 000 человек в месяц, можно рассчитать прогнозный доход от реализации онлайн-платформы по изучению иностранных языков. В первый год реализации продукта ожидается 10 000 подписчиков, в 2024 – 14 000, в 2025 – 18 000, а в 2026 – 11 000.

Так как организация-разработчик является резидентом Парка высоких технологий, то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле:

$$\Delta\Pi_{\text{ч}}^{\text{p}} = \Pi_{\text{опт}} \cdot N \cdot P_{\text{пр}} \quad (6.6)$$

где $\Pi_{\text{опт}}$ – прибыль, включаемая в цену программного средства, р.;
 N – количество копий (лицензий) ПО, реализуемое за месяц, шт.;
 $P_{\text{пр}}$ – рентабельность продаж копий (лицензий), %, $P_{\text{пр}} = 40\%$.

$$\Delta\Pi_{\text{ч}1}^{\text{p}} = 2,99 \cdot 15\,000 \cdot 0,4 = 17\,940 \text{ р.}$$

$$\Delta\Pi_{\text{ч}2}^{\text{p}} = 2,99 \cdot 14\,000 \cdot 0,4 = 16\,744 \text{ р.}$$

$$\Delta\Pi_{\text{ч}3}^{\text{p}} = 2,99 \cdot 13\,000 \cdot 0,4 = 15\,548 \text{ р.}$$

$$\Delta\Pi_{\text{ч}4}^{\text{p}} = 2,99 \cdot 11\,000 \cdot 0,4 = 13\,156 \text{ р.}$$

Таким образом, экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки за 4 года равен 57 408 р., что значительно превышает инвестиции в разработку онлайн-платформы для ее реализации на рынке.

6.4 Расчет показателей экономической эффективности разработки и реализации онлайн-платформы для изучения иностранных языков на рынке

Для оценки эффективности разработки продукта необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования соответствующего года, который определяется по формуле:

$$\alpha_t = (1 + d)^{t_p - t}, \quad (6.7)$$

где d – требуемая норма дисконта, (13%); t_p – расчётный год, к которому приводятся доходы и инвестиционные затраты, (1); t – номер года, результаты которого приводятся к расчетному.

Коэффициенты дисконтирования:

$$\alpha_1 = (1 + 0,13)^{1-1} = 1$$

$$\alpha_2 = (1 + 0,13)^{1-2} = 0,88$$

$$\alpha_3 = (1 + 0,13)^{1-3} = 0,78$$

$$\alpha_4 = (1 + 0,13)^{1-4} = 0,69$$

Расчёт показателей эффективности разработки продукта представлен в таблице 6.3.

Таблица 6.3 – Расчёт экономической эффективности разработки и реализации

Показатели	Едини- цы изме- рения	Расчетный период			
		2023	2024	2025	2026
РЕЗУЛЬТАТ					
Прирост чистой прибыли	р.	17 940	16 744	15 548	13 156
Коэффициент дисконтиро- вания	ед.	1	0,88	0,78	0,69
Дисконтированный ре- зультат	р.	17 940	14 734, 72	12 127,44	9 077, 64
Затраты на разработку программного продукта	р.	27 003, 74	—	—	—
Дисконтированные инве- стиции	р.	27 003, 74	—	—	—
Чистый дисконтирован- ный доход по годам	р.	-9 063, 74	14 734, 72	12 127, 44	9 077, 64
Чистый дисконтирован- ный доход с нарастающим итогом	р.	-9 063,74	5 670, 98	17 798, 42	26 876, 06

В ходе технико-экономического обоснования разрабатываемой онлайн-платформы для изучения иностранных языков с помощью карточек для запоминания произведён расчёт общих затрат на разработку программного обеспечения, выполнена оценка экономического эффекта от его реализации на массовом рынке, произведён расчёт эффективности инвестиций в разработку, а также показана его экономическая целесообразность.

Чистая прибыль от реализации программного продукта составит 26 876, 06 рублей.

Таким образом основываясь на произведённых расчётах и выполненных оценках, можно сделать вывод, что онлайн-платформа для изучения иностранных языков является экономически выгодным проектом.

ЗАКЛЮЧЕНИЕ

В ходе создания дипломного проекта была разработана онлайн-платформа для изучения иностранного языка с помощью чтения книг на иностранном языке и возможностью создания карточек с переводами неизвестных слов.

С использованием технологии React был разработан пользовательский интерфейс, который обеспечивает удобное и приятное взаимодействие пользователей с платформой. Интерфейс позволяет легко навигироваться по книгам, создавать и управлять карточками с переводами, а также отслеживать свой прогресс в изучении иностранного языка.

С помощью ASP.NET был разработан серверный компонент платформы, обеспечивающий обработку запросов, управление данными пользователей, аутентификацию и авторизацию. Это позволяет обеспечить безопасность и контроль доступа пользователей к функционалу платформы.

Платформа предоставляет доступ к библиотеке книг на иностранном языке разных уровней сложности. Пользователи могут загружать книги, соответствующие их уровню языковых навыков, и читать их непосредственно на платформе.

Одной из ключевых особенностей платформы является возможность создания карточек с переводами неизвестных слов. Это позволяет пользователям активно изучать новые слова и закреплять их, что способствует расширению словарного запаса и повышению уровня владения иностранным языком.

В результате разработки онлайн-платформы с использованием технологий React и ASP.NET была создана функциональная и удобная система.

Также в ходе работы над дипломным проектом рассмотрен экономический эффект от использования программного продукта. В результате расчётов подтвердилась целесообразность разработки.

В дальнейшем планируется разработать мобильное приложение, которое будет опираться на созданную онлайн-платформу, а также улучшить пользовательский интерфейс, добавив пользователю возможность полноэкранного режима, изменения размера шрифта и яркости экрана.

Таким образом в ходе написания данного диплома были использованы и структурированы знания, полученные за время обучения в университете.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Веб-приложение – Электронные данные – Режим доступа: <https://ru.wikipedia.org/wiki/Веб-приложение>
- [2] Visual Studio – Электронные данные – Режим доступа: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio
- [3] Visual Studio Code – Электронные данные – Режим доступа: https://ru.wikipedia.org/wiki/Visual_Studio_Code
- [4] Bookmate – Электронные данные – Режим доступа: <https://bookmate.com/books/yKvC2Q6l>
- [5] Feature-Sliced Design – Электронные данные – Режим доступа: <https://habr.com/ru/companies/inDrive/articles/693768/>
- [6] React Documentation – Электронные данные – Режим доступа: <https://ru.legacy.reactjs.org/>
- [7] Redux – Электронные данные – Режим доступа: <https://redux.js.org/tutorials/essentials/part-1-overview-concepts>
- [8] Entity Framework – Электронные данные – Режим доступа: <https://learn.microsoft.com/en-us/ef/core>
- [9] Система Лейтнера – Электронные данные – Режим доступа: https://ru.wikipedia.org/wiki/Система_Лейтнера
- [10] Алекс Банкс и Ив Порчелло, React и Redux функциональная разработка / Пер. с англ. / – СПб.: Издательство «Питер», 2019 – 336 с.
- [11] Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. / С. Макконнелл. – М. : Издательско-торговый дом «Русская редакция», 2010. – 896 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

```
using Enlightme.Core.Requests;
using Enlightme.Core.Responses;
using Enlightme.Dtos;
using Enlightme.Entities;
using Enlightme.Helpers;
using Enlightme.Models;
using Enlightme.Repositories;
using Enlightme.Services;
using Enlightme.Specifications;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace Enlightme.Controllers;

[Route("auth")]
[ApiController]
public class AuthController : Controller
{
    private readonly Repository<DataContext, User> userRepository;
    private readonly Repository<DataContext, RefreshToken> refreshToken-
Repository;
    private readonly JwtHelper jwtHelper;
    private readonly AuthService authService;

    public AuthController(
        Repository<DataContext, User> userRepository,
        Repository<DataContext, RefreshToken> refreshTokenRepository,
        JwtHelper jwtHelper,
        AuthService authService)
    {
        this.refreshTokenRepository = refreshTokenRepository;
        this.userRepository = userRepository;
        this.jwtHelper = jwtHelper;
        this.authService = authService;
    }

    [Route("register")]
    [HttpPost]
    public async Task<IActionResult> Register([FromForm] UserRegisterData
userRegisterData)
    {
        User user = await authService.RegisterUser(userRegisterData);

        return Created("success", user);
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login(UserLoginData userLoginData)
    {
        var user = await userRepository.GetFirstOrDefault(BaseSpecifica-
tion<User>.ByEmail(userLoginData.Email));

        if (user == null || !BCrypt.Net.BCrypt.Verify(userLoginData.Pass-
word, user.Password))
        {

```



```

        return BadRequest(new { message = "Invalid Credentials" });
    }

    AuthenticatedInfo authenticatedInfo = await authService.GetAuthenticatedInfo(user);
    Response.Cookies.Append("access", authenticatedInfo.AccessToken,
new CookieOptions
    {
        HttpOnly = true
    });

    return Ok(authenticatedInfo);
}

[Route("refresh")]
[HttpPost]
public async Task<IActionResult> Refresh([FromBody] RefreshRequestInfo refreshRequest)
{
    bool isValidRefreshToken = jwtHelper.Validate(refreshRequest.RefreshToken);

    if (!isValidRefreshToken)
    {
        return BadRequest(new ErrorResponse("Invalid token"));
    }

    var token = await refreshTokenRepository.GetFirstOrDefault(
        BaseSpecification<RefreshToken>.ByToken(refreshRequest.RefreshToken));

    if (token == null)
    {
        return NotFound(new ErrorResponse("Invalid token"));
    }

    User user = await userRepository.GetFirstOrDefault(
        BaseSpecification<User>.ById(token.UserId));

    await refreshTokenRepository.Delete(token);

    if (user == null)
    {
        return NotFound(new ErrorResponse("Invalid token"));
    }

    AuthenticatedInfo authenticatedInfo = await authService.GetAuthenticatedInfo(user);

    await refreshTokenRepository.Create(new RefreshToken()
    {
        Token = authenticatedInfo.RefreshToken,
        UserId = user.Id
    });

    return Ok(authenticatedInfo);
}

[Authorize]
[HttpDelete("logout")]
public async Task<IActionResult> Logout()

```

```

        {
            Response.Cookies.Delete("access");

            var rawUserId = HttpContext.User.FindFirstValue("id");

            if(!int.TryParse(rawUserId, out int userId))
            {
                return Unauthorized();
            }

            await refreshTokenRepository.Delete(BaseSpecification<RefreshToken>.ByUserId(userId));

            return Ok(new SuccessResponse("Success"));
        }
    }

    using Enlightme.Dtos;
    using Enlightme.Entities;
    using Enlightme.Helpers;
    using Enlightme.Models;
    using Enlightme.Repositories;

    namespace Enlightme.Services;

    public class AuthService
    {
        private readonly Repository<DataContext, User> userRepository;
        private readonly Repository<DataContext, RefreshToken> refreshTokenRepository;
        private readonly JwtHelper jwtHelper;

        public AuthService(
            Repository<DataContext, User> userRepository,
            Repository<DataContext, RefreshToken> refreshTokenRepository,
            JwtHelper jwtHelper)
        {
            this.refreshTokenRepository = refreshTokenRepository;
            this.userRepository = userRepository;
            this.jwtHelper = jwtHelper;
        }

        public async Task<User> RegisterUser(UserRegisterData userRegisterData)
        {
            var user = new User
            {
                FirstName = userRegisterData.FirstName,
                LastName = userRegisterData.LastName,
                Email = userRegisterData.Email,
                BirthDate = userRegisterData.BirthDate,
                ShouldSendNotifications = true,
                Password = BCrypt.Net.BCrypt.HashPassword(userRegisterData.Password)
            };

            if (userRegisterData.Image != null)
            {
                byte[]? imageData = null;
            }
        }
    }

```

```

        using (var binaryReader = new BinaryReader(userRegister-
Data.Image.OpenReadStream()))
        {
            imageData = binaryReader.ReadBytes((int)userRegister-
Data.Image.Length);
        }

        user.Image = imageData;
    }

    return await userRepository.Create(user);
}

public async Task<AuthenticatedInfo> GetAuthenticatedInfo(User user)
{
    var accessToken = jwtHelper.GenerateAccessToken(user);
    var refreshToken = jwtHelper.GenerateRefreshToken();

    RefreshToken refreshTokenDto = new RefreshToken()
    {
        Token = refreshToken,
        UserId = user.Id,
    };

    await refreshTokenRepository.Create(refreshTokenDto);

    return (new AuthenticatedInfo
    {
        AccessToken = accessToken,
        RefreshToken = refreshToken
    });
}

}

using Enlightme.Models;
using Microsoft.IdentityModel.Tokens;
using Microsoft.Extensions.Options;
using System.IdentityModel.Tokens.Jwt;
using System.Text;
using System.Security.Claims;
using Enlightme.Entities;

namespace Enlightme.Helpers
{
    public class JwtHelper
    {
        private readonly AuthenticationConfiguration authenticationCon-
figuration;

        public JwtHelper(IOptions<AuthenticationConfiguration> options)
        {
            this.authenticationConfiguration = options.Value;
        }

        public string GenerateAccessToken(User user)
        {
            List<Claim> claims = new List<Claim>()
            {
                new Claim("id", user.Id.ToString()),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Name, user.FirstName)
            }
        }
    }
}

```

```

        };

        var token = Generate(
            authenticationConfiguration.AccessTokenSecret,
            authenticationConfiguration.Issuer,
            authenticationConfiguration.Audience,
            authenticationConfiguration.AccessTokenExpira-
tionMinutes,
            claims);

        return token;
    }

    public string GenerateRefreshToken()
    {
        var token = Generate(
            authenticationConfiguration.RefreshTokenSecret,
            authenticationConfiguration.Issuer,
            authenticationConfiguration.Audience,
            authenticationConfiguration.RefreshTokenExpira-
tionMinutes);

        return token;
    }

    public bool Validate(string refreshToken)
    {
        try
        {
            Verify(refreshToken, true);

            return true;
        }
        catch (Exception)
        {
            return false;
        }
    }

    public JwtSecurityToken Verify(string jwt, bool isRefreshToken =
false)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = isRefreshToken ? authenticationConfiguration.Re-
freshTokenSecret : authenticationConfiguration.AccessTokenSecret;

        tokenHandler.ValidateToken(jwt, new TokenValidationParame-
ters
        {
            IssuerSigningKey = new SymmetricSecurityKey(Encod-
ing.ASCII.GetBytes(key)),
            ValidIssuer = authenticationConfiguration.Issuer,
            ValidAudience = authenticationConfiguration.Audience,
            ValidateIssuerSigningKey = true,
            ValidateIssuer = false,
            ValidateAudience = false,
            ClockSkew = TimeSpan.Zero
        }, out SecurityToken securityToken);

        return (JwtSecurityToken)securityToken;
    }

```

```

        private string Generate(
            string secretKey,
            string issuer,
            string audience,
            double expirationMinutes,
            IEnumerable<Claim>? claims = null)
        {
            var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
            var credentials = new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256Signature);
            var header = new JwtHeader(credentials);

            var payload = new JwtPayload(issuer, audience, claims, DateTime.UtcNow, DateTime.UtcNow.AddMinutes(expirationMinutes));
            var securityToken = new JwtSecurityToken(header, payload);

            return new JwtSecurityTokenHandler().WriteToken(securityToken);
        }
    }

    using Enlightme.Dtos;
    using Enlightme.Entities;
    using Enlightme.Services;
    using Microsoft.AspNetCore.Authorization;
    using Microsoft.AspNetCore.Mvc;
    using System.Security.Claims;

    namespace Enlightme.Controllers;

    [Authorize]
    [Route("card")]
    [ApiController]
    public class CardController : Controller
    {
        private readonly CardService cardService;

        public CardController(CardService cardService)
        {
            this.cardService = cardService;
        }

        [Route("create")]
        [HttpPost]
        public async Task<IActionResult> Create(CardData cardData)
        {
            var rawUserId = HttpContext.User.FindFirstValue("id");

            if (!int.TryParse(rawUserId, out int userId))
            {
                return Unauthorized();
            }

            Card card = await cardService.CreateCard(cardData, userId);

            return Ok(card);
        }
    }

```

```

[Route("notifications")]
[HttpGet]
public async Task<IActionResult> GetNotifications()
{
    var rawUserId = HttpContext.User.FindFirstValue("id");

    if (!int.TryParse(rawUserId, out int userId))
    {
        return Unauthorized();
    }

    int amount = await cardService.GetNotificationAmount(userId);

    return Ok(amount);
}

[Route("cards/learn/")]
[HttpGet]
public async Task<IActionResult> GetCards()
{
    var rawUserId = HttpContext.User.FindFirstValue("id");

    if (!int.TryParse(rawUserId, out int userId))
    {
        return Unauthorized();
    }

    IReadOnlyList<Card> cards = await cardService.GetCards(userId);

    return Ok(cards);
}

[Route("cards")]
[HttpGet]
public async Task<IActionResult> GetAllCards()
{
    var rawUserId = HttpContext.User.FindFirstValue("id");

    if (!int.TryParse(rawUserId, out int userId))
    {
        return Unauthorized();
    }

    IReadOnlyList<Card> cards = await cardService.GetAllCards(userId);

    return Ok(cards);
}

[Route("delete/{cardId:int}")]
[HttpDelete]
public async Task<IActionResult> Delete(int cardId)
{
    bool result = await cardService.DeleteCard(cardId);

    return Ok(result);
}

[Route("{id:int}/update")]
[HttpPost]

```

```

        public async Task<IActionResult> UpdateCard(CardUpdateData cardUpdateData)
        {
            var rawUserId = HttpContext.User.FindFirstValue("id");

            if (!int.TryParse(rawUserId, out int userId))
            {
                return Unauthorized();
            }

            var card = await cardService.UpdateCard(cardUpdateData);

            return Ok(card);
        }
    }

    using Enlightme.Interfaces;
    using Enlightme.Specifications;
    using Microsoft.EntityFrameworkCore;
    using System.Linq.Expressions;

    namespace Enlightme.Repositories;

    public class Repository<TContext, TEntity>
        where TContext : DbContext
        where TEntity : class, IHasId
    {
        private readonly TContext dataContext;
        protected readonly DbSet<TEntity> DbSet;

        public Repository(TContext dataContext)
        {
            this.dataContext = dataContext;
            DbSet = dataContext.Set<TEntity>();
        }

        public async Task<TEntity> Create(TEntity entity)
        {
            await dataContext.AddAsync<TEntity>(entity);
            await dataContext.SaveChangesAsync();

            return entity;
        }

        public async Task Update(TEntity newEntity)
        {
            TEntity oldentity = await GetFirstOrDefault(new Specification<TEntity>(e => e.Id == newEntity.Id));
            oldentity = newEntity;

            await dataContext.SaveChangesAsync();
        }

        public async Task Delete(TEntity entity)
        {
            dataContext.Remove(entity);
            await dataContext.SaveChangesAsync();
        }

        public async Task Delete(Specification<TEntity> specification)
        {

```

```

        await DbSet.Where(specification.ToExpression()).ExecuteDeleteAsync();

        await dbContext.SaveChangesAsync();
    }

    protected virtual IQueryable<TEntity> GetDbSet()
    {
        return DbSet;
    }

    protected virtual IQueryable<TEntity> GetSpecifiedQuery(Specification<TEntity>? specification = null, IncludedPropertyCollection<TEntity> includes = null)
    {
        IQueryable<TEntity> query = (IQueryable<TEntity>)GetDbSet();
        Expression<Func<TEntity, bool>> expression = specification?.ToExpression();
        if (expression != null)
        {
            query = query.Where(expression);
        }

        if (includes != null)
        {
            query = includes.Aggregate(query, (current, include) => current.Include(include.AsPath()));
        }

        return query;
    }

    public virtual async Task<int> Count(Specification<TEntity> specification)
    {
        return await GetSpecifiedQuery(specification).CountAsync();
    }

    public async Task<TEntity> GetFirstOrDefault(Specification<TEntity> specification, IncludedPropertyCollection<TEntity> includes = null)
    {
        IQueryable<TEntity> query = GetSpecifiedQuery(specification, includes);

        return await query.FirstOrDefaultAsync();
    }

    public async Task<IReadOnlyList<TEntity>> GetList(Specification<TEntity>? specification = null, IncludedPropertyCollection<TEntity> includes = null)
    {
        IQueryable<TEntity> query = GetSpecifiedQuery(specification, includes);

        return await query.ToListAsync();
    }
}

using System.Linq.Expressions;

namespace Enlightme.Specifications;

```



```

public class Specification<T>
{
    private readonly Expression<Func<T, bool>> expression;

    private Func<T, bool> compiledDelegate;
    private Func<T, bool> CompiledDelegate => compiledDelegate ??= ex-
pression.Compile();

    public Specification(Expression<Func<T, bool>> expression)
    {
        this.expression = expression;
    }

    public Expression<Func<T, bool>> ToExpression()
    {
        return expression;
    }

    public bool IsSatisfiedBy(T value)
    {
        return CompiledDelegate(value);
    }
}

using Enlightme.Dtos;
using Enlightme.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace Enlightme.Controllers;

[Route("api")]
[ApiController]
public class UserController : Controller
{
    private readonly UserService userService;

    public UserController(UserService userService)
    {
        this.userService = userService;
    }

    [Authorize]
    [HttpGet("user")]
    public async Task<IActionResult> User()
    {
        try
        {
            var jwt = Request.Cookies["access"];

            var user = await userService.GetUser(jwt);

            return Ok(user);
        }
        catch (Exception _)
        {
            return Unauthorized();
        }
    }
}

```

```

[HttpGet("image")]
public async Task<IActionResult> GetUserImage()
{
    try
    {
        var jwt = Request.Cookies["access"];

        var user = await userService.GetUser(jwt);

        return Ok(user.Image);
    }
    catch (Exception _)
    {
        return Unauthorized();
    }
}

[HttpGet("settings")]
public async Task<IActionResult> GetUserSettings()
{
    try
    {
        var jwt = Request.Cookies["access"];

        var user = await userService.GetUser(jwt);

        return Ok(user);
    }
    catch (Exception _)
    {
        return Unauthorized();
    }
}

[HttpPost("settings")]
public async Task<IActionResult> SetUserSettings([FromForm] UserSettingData userSettingData)
{
    try
    {
        var user = await userService.SetUserSettings(userSettingData);

        return Ok(user);
    }
    catch (Exception _)
    {
        return Unauthorized();
    }
}

```

```

using Enlightme.Core.Responses;
using Enlightme.Dtos;
using Enlightme.Entities;
using Enlightme.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

```

```

namespace Enlightme.Controllers;

```

```

[Route("book")]
[ApiController]
public class BookController : Controller
{
    private readonly BookService bookService;

    public BookController(BookService bookService)
    {
        this.bookService = bookService;
    }

    [Route("library")]
    [HttpGet]
    public async Task<IActionResult> GetBooks()
    {
        IReadOnlyList<Book> books = await bookService.GetBooks(null);

        return Created("success", books);
    }

    [Authorize]
    [Route("books")]
    [HttpGet]
    public async Task<IActionResult> GetUserBooks()
    {
        var rawUserId = HttpContext.User.FindFirstValue("id");

        if (!int.TryParse(rawUserId, out int userId))
        {
            return Unauthorized();
        }

        IReadOnlyList<Book> books = await bookService.GetBooks(userId);

        return Ok(books);
    }

    [Route("{bookId:int}")]
    [HttpGet]
    public async Task<IActionResult> GetBook(int bookId)
    {
        Book book = await bookService.GetBook(bookId);

        return Ok(book);
    }

    [Route("common-info")]
    [HttpGet]
    public async Task<IActionResult> GetGenres()
    {
        IReadOnlyList<Genre> genres = await bookService.GetGenres();
        IReadOnlyList<Language> languages = await bookService.GetLanguages();

        var data = new CommonData(genres, languages);

        return Ok(data);
    }

    [Route("languages")]

```

```

        [HttpGet]
        public async Task<IActionResult> GetLanguages()
        {
            IReadOnlyList<Language> languages = await bookService.GetLanguages();

            return Ok(languages);
        }

        [Authorize]
        [Route("create")]
        [HttpPost]
        public async Task<IActionResult> Create([FromForm] BookData bookData)
        {
            var rawUserId = HttpContext.User.FindFirstValue("id");

            if (!int.TryParse(rawUserId, out int userId))
            {
                return Unauthorized();
            }

            Book book = await bookService.CreateBook(bookData, userId);

            return Created("success", book);
        }

        [Authorize]
        [Route("delete/{bookId:int}")]
        [HttpDelete]
        public async Task<IActionResult> Delete(int bookId)
        {
            var rawUserId = HttpContext.User.FindFirstValue("id");

            if (!int.TryParse(rawUserId, out int userId))
            {
                return Unauthorized();
            }

            bool isDeleted = await bookService.DeleteBook(bookId);

            return isDeleted ? Ok(isDeleted) : BadRequest(new ErrorResponse("Can't delete the book."));
        }
    }

    using Enlightme.Dtos;
    using Enlightme.Entities;
    using Enlightme.Helpers;
    using Enlightme.Repositories;
    using Enlightme.Specifications;

    namespace Enlightme.Services;

    public class BookService
    {
        private readonly Repository<DataContext, Book> bookRepository;
        private readonly Repository<DataContext, Genre> genreRepository;
        private readonly Repository<DataContext, Language> languagesRepository;

        private readonly FileHelper fileHelper;

        public BookService(

```

```

        FileHelper fileHelper,
        Repository<DataContext, Book> bookRepository,
        Repository<DataContext, Genre> genreRepository,
        Repository<DataContext, Language> languagesRepository)
    {
        this.bookRepository = bookRepository;
        this.fileHelper = fileHelper;
        this.genreRepository = genreRepository;
        this.languagesRepository = languagesRepository;
    }

    public async Task<IReadOnlyList<Genre>> GetGenres()
    {
        var genres = await genreRepository.GetList();

        return genres;
    }

    public async Task<IReadOnlyList<Language>> GetLanguages()
    {
        var languages = await languagesRepository.GetList();

        return languages;
    }

    public async Task<IReadOnlyList<Book>> GetBooks(int? userId = null)
    {
        var includes = new IncludedPropertyCollection<Book>()
        {
            b => b.Genre
        };

        if (userId != null)
        {
            Specification<Book> specification = new(b => b.UserId ==
userId);

            return await bookRepository.GetList(specification, in-
cludes);
        }

        return await bookRepository.GetList(includes: includes);
    }

    public async Task<Book> GetBook(int bookId)
    {
        Specification<Book> specification = new(b => b.Id == bookId);
        var includes = new IncludedPropertyCollection<Book>()
        {
            b => b.Genre,
            b => b.Language
        };

        return await bookRepository.GetFirstOrDefault(specification, in-
cludes);
    }

    public async Task<Book> CreateBook(BookData bookData, int userId)
    {
        var book = new Book
        {

```

```

        Title = bookData.Title,
        Description = bookData.Description,
        Author = bookData.Author,
        UserId = userId,
        GenreId = bookData.Genre,
        LanguageId = bookData.Language,
        Content = fileHelper.GetFileBytes(bookData.Book)
    };

    if (bookData.Cover != null )
    {
        book.Cover = fileHelper.GetFileBytes(bookData.Cover);
    }

    return await bookRepository.Create(book);
}

public async Task<bool> DeleteBook(int bookId)
{
    try
    {
        Specification<Book> specification = new(b => b.Id == bookId);

        await bookRepository.Delete(specification);

        return true;
    }
    catch (Exception)
    {
        return false;
    }
}
}

using Enlightme.Dtos;
using Enlightme.Entities;
using Enlightme.Helpers;
using Enlightme.Repositories;
using Enlightme.Specifications;
using System.IdentityModel.Tokens.Jwt;

namespace Enlightme.Services;

public class UserService
{
    private readonly Repository<DataContext, User> userRepository;
    private readonly JwtHelper jwtHelper;

    public UserService(Repository<DataContext, User> userRepository,
JwtHelper jwtHelper)
    {
        this.userRepository = userRepository;
        this.jwtHelper = jwtHelper;
    }

    public async Task<User> GetUser(string accessToken)
    {
        var token = jwtHelper.Verify(accessToken);

        int userId = int.Parse(token.Claims.FirstOrDefault(c => c.Type ==
"id").Value);

```

```

        var user = await userRepository.GetFirstOrDefault(BaseSpecification<User>.ById(userId));

        return user;
    }

    public async Task<User> GetUserSettings(string accessToken)
    {
        var token = jwtHelper.Verify(accessToken);

        int userId = int.Parse(token.Claims.FirstOrDefault(c => c.Type ==
"id").Value);

        var user = await userRepository.GetFirstOrDefault(BaseSpecification<User>.ById(userId));

        var userSettingsData = new UserSettingData
        {
            FirstName = user.FirstName,
            LastName = user.LastName,
            ShouldSendNotifications = user.ShouldSendNotifications,
            BirthDate = user.BirthDate
        };

        return user;
    }

    public async Task<User> SetUserSettings(UserSettingData userSetting-
Data)
    {
        var user = await userRepository.GetFirstOrDefault(BaseSpecification<User>.ById(userSettingData.UserId));

        user.FirstName = userSettingData.FirstName;
        user.LastName = userSettingData.LastName;
        user.ShouldSendNotifications = userSettingData.ShouldSendNotifi-
cations;
        user.BirthDate = userSettingData.BirthDate;

        if (userSettingData.Image != null)
        {
            byte[]? imageData = null;

            using (var binaryReader = new BinaryReader(userSetting-
Data.Image.OpenReadStream()))
            {
                imageData = binaryReader.ReadBytes((int)userSetting-
Data.Image.Length);
            }

            user.Image = imageData;
        }

        await userRepository.Update(user);

        return user;
    }
}

import React, {useCallback, useState} from "react";

```

```

import { Navigate } from 'react-router-dom';

import BasePage from "src/components/base/BasePage/basePage";
import Input from 'components/base/Input/input';
import PageName from 'src/features/common/components/PageName/pageName';
import PagesMenu from "features/common/components/PagesMenu/pagesMenu";
import requestHelper from "src/helpers/requestHelper";
import serviceUrls from "src/constants/serviceUrls";
import routes from "src/constants/routes";
import defaultImg from "src/static/images/defaultImage.png"

import './signUpPage.scss';

const defaultImage = {
  imageSrc: defaultImg,
  imageFile: null
};

export default function SignUpPage() {
  const [firstName, setFirstName] = useState('');
  const [lastName, setLastName] = useState('');
  const [image, setImage] = useState(defaultImage);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [repeatPassword, setRepeatPassword] = useState('');
  const [birthDate, setBirthDate] = useState('');
  const [register, setRegister] = useState(false);
  const [warning, setWarning] = useState({
    email: [],
    password: [],
    username: [],
    birthDate: [],
  });

  const onFirstNameChange = useCallback(e => {
    setFirstName(e.target.value);
  }, []);

  const onLastNameChange = useCallback(e => {
    setLastName(e.target.value);
  }, []);

  const onEmailChange = useCallback(e => {
    setWarning({...warning, email: []})
    setEmail(e.target.value);
  }, []);

  const onImageChange = useCallback(e => {
    if (e.target.files && e.target.files[0])
    {
      let imageFile = e.target.files[0];
      const reader = new FileReader();
      reader.onload = x => {
        setImage({
          imageFile,
          imageSrc: x.target.result,
        })
      }
      reader.readAsDataURL(imageFile);
    }
    else {

```



```

        setImage({
          imageFile: null,
          imageSrc: defaultImg
        })
      }
    }, []);

    const onChange = useCallback(e => {
      setWarning({...warning, birthDate: []})
      setBirthDate(e.target.value);
    }, []);

    const onPasswordChange = useCallback(e => {
      setWarning({...warning, password: []})
      setPassword(e.target.value);
    }, []);

    const onRepeatPasswordChange = useCallback(e => {
      setRepeatPassword(e.target.value);
      setWarning({
        ...warning, password: [
          e.target.value !== password
            ? 'You entered two different passwords. Please try again.'
            : ''
        ]
      });
    }, [password]);

    const onSubmit = useCallback(async e => {
      e.preventDefault();
      if (repeatPassword === password) {
        setWarning('');
        const userImage = image.imageFile;
        const user = {
          firstName,
          lastName,
          password,
          email,
          image: userImage,
          birthDate
        };

        const formData = new FormData();
        for (const key in user) {
          formData.append(key, user[key]);
        }

        const response = await requestHelper.get(serviceUrls.registration, {
          method: 'POST',
          body: formData
        });

        if (response) {
          setRegister(true);
        } else {
          setWarning(
            {...warning, password: ['Passwords differ one from another']}
          );
        }
      }
    }, [
      firstName,
      lastName,
      email,
      password,
      repeatPassword,
      birthDate,
      image
    ]);
  }, [
    firstName,
    lastName,
    email,
    password,
    repeatPassword,
    birthDate,
    image
  ]);
}

```

```

    }
    }, [birthDate, repeatPassword, email, image, lastName, password]);

    if (register) {
        return <Navigate to={routes.LOG_IN} />;
    }

    return (
        <div className="page-container">
            <BasePage isLoginPage>
                <PageName title={'Sign Up'} />
                <section className="sign-up-page__section">
                    <form className="sign-up-page__form" onSubmit={on-
FormSubmit}>
                        <div className="sign-up-page__form-fields">
                            <div className="sign-up-page__form-column">
                                <Input type={'text'} placeholder={'first
name'} onChange={onFirstNameChange} withBorder />
                                <p className="" style={{width:
200}}>{warning.username}</p>
                                <Input type={'date'} placeholder={'birth
day'} lang={'en-UK'} onChange={onDateChange} withBorder />
                                <p className="" style={{width:
200}}>{warning.birthDate}</p>
                                <Input type={'password'} place-
holder={'password'} onChange={onPasswordChange} withBorder />
                                <Input type={'password'} place-
holder={'repeat password'} onChange={onRepeatPasswordChange} withBorder />
                            </div>
                            <div className="sign-up-page__form-column">
                                <Input type={'text'} placeholder={'last
name'} onChange={onLastNameChange} withBorder />
                                <Input type={'email'} place-
holder={'email'} onChange={onEmailChange} withBorder />
                                <p className="" style={{width:
200}}>{warning.email}</p>
                                <div className="sign-up-page__photo-con-
tainer">
                                    <img src={image.imageSrc} class-
Name="sign-up-page__photo"/>
                                </div>
                                <div className="sign-up-page__photo-in-
put">
                                    <input id="sign-up-photo"
type={'file'} onChange={onImageChange} />
                                    <label htmlFor="sign-up-
photo">choose photo</label>
                                </div>
                            </div>
                        </div>
                        <p className="" style={{width: 400}}>{warn-
ing.password}</p>
                        <div className="sign-up-page__button-container">
                            <button type="submit" className="sign-up-
page__button">Sign up</button>
                        </div>
                    </form>
                </section>
            </BasePage>
        </div>
    )

```

```
}
```

```
import React, { useState, useEffect, useCallback } from "react";
import { Link, useLocation, useNavigate } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";

import BasePage from "src/components/base/BasePage/basePage";
import PageName from "features/common/components/PageName/pageName";
import urlHelper from "src/helpers/urlHelper";
import serviceUrls from "src/consts/serviceUrls";
import requestHelper from "src/helpers/requestHelper";
import routes from "src/consts/routes";

import bookCover from 'src/static/images/cover.png';

import './bookInfoPage.scss';

export default function BookInfoPage() {
  const dispatch = useDispatch();
  const location = useLocation();
  const bookId = +location.pathname.slice(6);
  const [book, setBook] = useState();
  const isAuthenticated = useSelector(state => state.users.isAuthenticated);

  useEffect(() => {
    if (bookId) {
      dispatch(async () => {
        const url = urlHelper.getUrlByTemplate(
          serviceUrls.getUserBookId, {id: bookId}
        );

        const data = await requestHelper.get(
          url, {
            method: 'GET',
            headers: {
              'Content-Type': 'application/json',
            }
          }, isAuthenticated
        );

        if (data) {
          setBook(data);
        }
      })
    }
  }, [bookId, dispatch, isAuthenticated, setBook]);

  const formHandler = useCallback(async (e) => {
    e.preventDefault();

    if (!isAuthenticated) {
      window.location.replace(routes.LOG_IN);
    }

    const data = await requestHelper.get(
      url, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        }
      }
    )
  })
}
```

```

        }, isAuth
    );

    if (data.url) {
        window.location.replace(data.url)
    }
}, [isAuth]);

const onDeleteButtonClick = useCallback(async () => {
    const url = urlHelper.getUrlByTemplate(
        serviceUrls.deleteBook, { bookId }
    );

    const data = await requestHelper.get(
        url, {
            method: 'DELETE',
            headers: {
                'Content-Type': 'application/json',
            }
        }, isAuth, true
    );

    if (data) {
        window.location.replace(routes.USER_BOOKS);
    }
}, [bookId]);

if (!book) {
    return <div></div>;
}

return (
    <BasePage>
        {
            book && (<
                <PageName title={book.title} />
                <div className="book-info-page__author-container">
                    <h2 className="book-info-page__author">{
                        book.author
                    }</h2>
                </div>
                <section className="book-info-page__book-section">
                    <div
                        className="book-info-page__book-visual-
info">
                        <div className="book-info-page__book-cover-
container">
                            <img
                                className="book-info-page__book-
cover" src={book.cover ?? bookCover} alt='cover' />
                            </div>
                            {
                                book.id && (
                                    <div
                                        className="book-info-
page__price-info">
                                            <Link
                                                className="book-info-
page__link" to={urlHelper.getUrlByTemplate(routes.USER_BOOK_CARDS, { id:
book.id })}>
                                                <button
                                                    className="book-
info-page__button">cards</button>
                                            </Link>
                                            <button
                                                className="book-info-
page__button" onClick={onDeleteButtonClick}>delete the book</button>

```

```

        </div>
    )
}
<div></div>
</div>
<div className="book-info-page__book-text-info">
    <div className="book-info-page__book-main-
info-container">
        {
            book.genre &&
            <span className="book-info-
page__book-main-info">
                Genre: {book.genre.type}<br
/>
            </span>
        }
        {
            book.language &&
            <span className="book-info-
page__book-main-info">
                Language: {book.lan-
guage.languageName}<br />
            </span>
        }
        {
            book.publicationDate &&
            <span className="book-info-
page__book-main-info">
                Publication date: {book.pub-
lication_date}
            </span>
        }
    </div>
    <div className="book-info-page__book-de-
scription-container">
        <span className="book-info-page__book-
description">{book.description ? book.description : 'There is no descrip-
tion.'}</span>
    </div>
</div>
</section>
{
    book.id && (
        <div className="book-info-page__read-but-
ton">
            <Link className="book-info-page__read-
button" to={urlHelper.getUrlByTemplate(routes.READ_BOOK, { id: book.id })}>
                <button className="book-info-
page__button">read</button>
            </Link>
        </div>
    )
}
</>
}
</BasePage>
)
}

```