

## Содержание

Введение .....	5
1 Постановка задачи и обзор аналогов .....	6
1.1 Постановка задачи .....	6
1.2 Основные сведения по теме «Веб-приложение» .....	6
1.3 Обзор аналогичных решений.....	7
1.3.1 Веб-приложение Duolingo.com.....	7
1.3.2 Веб-приложение Poliglot16.ru.....	8
1.2.3 Веб-приложение Puzzel-english.com .....	8
1.4 Анализ прототипов .....	9
2 Проектирование веб-приложения .....	10
2.1 Функциональные возможности веб-приложения .....	10
2.2 Логическая схема базы данных .....	13
2.3 Архитектура веб-приложения .....	18
2.4 Вывод по разделу .....	20
3 Реализация веб-приложения .....	21
3.1 Программная платформа Node.js .....	21
3.2 СУБД PostgreSQL .....	21
3.3 Prisma ORM .....	21
3.4 Библиотеки и фреймворки .....	22
3.5 Структура серверной части.....	24
3.6 Сторонние сервисы .....	27
3.7 Реализация функций для роли «Гость».....	27
3.7.1 Регистрация .....	27
3.7.2 Авторизация .....	27
3.8 Реализация функций для роли «Пользователь» .....	28
3.8.1 Просмотр своих словарей .....	28
3.8.2 Добавление словаря .....	29
3.8.3 Удаление словаря.....	29
3.8.4 Изменение своего словаря .....	30
3.8.5 Добавление своего слова.....	30
3.8.6 Изменение своего слова .....	31
3.8.7 Удаление своего слова .....	31
3.8.8 Импорт слов.....	32
3.8.9 Экспорт слов.....	32
3.8.10 Просмотр словарей администратора .....	33
3.8.11 Просмотр статистики.....	34
3.8.12 Решение заданий .....	34
3.8.13 Просмотр уроков.....	35
3.8.14 Просмотр настроек аккаунта .....	35
3.8.15 Смена пароля .....	36
3.8.16 Смена количества правильных ответов.....	36
3.8.17 Смена режима изучения грамматики.....	37
3.8.18 Смена количества изучаемых слов.....	38

3.8.19	Смена режима изучения лексикона .....	39
3.8.20	Спросить ответ у искусственного интеллекта .....	40
3.8.21	Просмотр слова .....	40
3.8.22	Перевод слов в соревновательном режиме .....	41
3.9	Реализация функций для роли «Администратор» .....	43
3.9.1	Изменение урока .....	43
3.9.2	Добавление урока.....	43
3.9.3	Удаление урока .....	44
3.9.4	Добавление задания .....	44
3.9.5	Изменение задания .....	45
3.9.6	Удаление задания.....	45
3.9.7	Просмотр заданий .....	46
3.10	Структура клиентской части .....	46
3.11	Вывод по разделу .....	49
4	Тестирование программного продукта.....	50
4.1	Функциональное тестирование для роли Пользователя.....	50
4.2	Функциональное тестирование для роли Администратора .....	53
4.3	Функциональное тестирование для роли Гостя .....	56
4.4	Вывод по разделу .....	56
5	Руководство пользователя .....	57
5.1	Регистрация .....	57
5.2	Авторизация .....	57
5.3	Просмотр своих словарей .....	58
5.4	Добавление словаря.....	58
5.5	Удаление своего словаря.....	59
5.6	Изменение своего словаря .....	60
5.7	Добавление своего слова.....	61
5.8	Изменение своего слова .....	62
5.9	Удаление своего слова.....	63
5.10	Импорт слов.....	64
5.11	Экспорт слов.....	65
5.12	Просмотр словарей Администратора .....	67
5.13	Просмотр статистики.....	67
5.14	Решение заданий .....	68
5.15	Просмотр уроков.....	69
5.16	Просмотр настроек аккаунта .....	70
5.17	Смена пароля .....	70
5.18	Смена количества правильных ответов.....	71
5.19	Смена режима изучения грамматики.....	71
5.20	Смена количества изучаемых слов .....	72
5.21	Смена режима изучения лексикона .....	73
5.22	Просмотр слова .....	74
5.24	Спросить ответ у искусственного интеллекта .....	75
5.25	Перевод слов в соревновательном режиме .....	75
5.27	Изменение урока .....	76

5.28 Добавление урока.....	77
5.30 Добавление задания.....	78
5.31 Изменение задания .....	79
5.32 Удаление задания.....	80
5.34 Просмотр заданий .....	81
Заключение .....	82
Список использованных источников.....	83
Приложение А .....	86
Приложение Б.....	90
Приложение В .....	92
Приложение Г.....	94
<b>Приложение Д.....</b>	<b>97</b>
<b>Приложение Е.....</b>	<b>101</b>
<b>Приложение Ж.....</b>	<b>106</b>

## Введение

Целью данного курсового проекта является разработка веб-приложения для изучения испанского языка. Оно будет позволять пользователям улучшать свои навыки и знания в любое удобное время и в любом месте. В рамках проекта будет рассмотрено несколько аспектов, необходимых для создания и успешного функционирования веб-приложения.

Для достижения цели были поставлены следующие задачи:

1. Провести анализ существующих интернет-магазинов электроники и определить ключевые требования к разрабатываемому веб-приложению (раздел 1);
2. Разработать архитектуру веб-приложения (раздел 2);
3. Реализовать функционал веб-приложения с учетом поставленных требований (раздел 3);
4. Провести тестирование для выявления и устранения ошибок, а также для проверки соответствия требованиям (раздел 4);
5. Разработать руководство пользователя веб-приложения (раздел 5)

Целевая аудитория веб-приложения — люди с доходом от среднего и выше среднего, которые ищут надежную электронику для повседневной жизни, работы, учебы и досуга.

В качестве программной платформы было решено использовать Node.js [1]

## 1 Постановка задачи и обзор аналогов

В ходе выполнения курсового проекта были проанализированы аналоги приложения для самостоятельного изучения испанского языка, выявлены их отличительные особенности и поставлены задачи курсового проекта.

### 1.1 Постановка задачи

Анализ веб-приложений, в основном, сводится к удобству пользовательского интерфейса, а также реализации всех функций необходимых пользователю. В данном разделе был произведен обзор аналогов приложений для самостоятельного изучения испанского языка, рассмотрены их плюсы и минусы. Каждый аналог был подробно описан, и было показано, для чего предназначено то или иное программное обеспечение.

Исходя из сделанных выводов при анализе аналоговых приложений было решено реализовать следующие ключевые функциональные возможности:

- регистрация и авторизация пользователя;
- реализация ролей Гостя, Пользователя и Администратора;
- тренировка грамматики и расширение словарного запаса;
- база грамматических заданий и словарей;
- настройки обучения;
- статистика пользователя.

Программа должна быть предназначена для различной аудитории пользователей. Это значит, что приложение должно быть простое и иметь доступный дизайн.

Все эти пункты были учтены при выполнении курсового проектирования.

### 1.2 Основные сведения по теме «Веб-приложение»

Веб-приложение представляет собой веб-сайт, на котором размещены страницы с частично либо полностью несформированным содержанием. Окончательное содержание формируется только после того, как посетитель сайта запросит страницу с веб-сервера. В связи с тем, что окончательное содержание страницы зависит от запроса, созданного на основе действий посетителя, такая страница называется динамической.

Спектр использования веб-приложений очень широк. Использование веб-приложений приносит определенную пользу как посетителям веб-сайтов, так и их разработчикам. Например, посетители могут быстро и легко находить требуемую информацию на веб-сайтах с большим объемом информации, а разработчики сохранять и анализировать данные, полученные от посетителей сайта.

Долгое время использовался метод, при котором данные, введенные в *HTML*-формы, отсылались для обработки *CGI*-приложениям или специально назначенным работникам в виде сообщений электронной почты. Веб-приложение позволяет сохранять данные непосредственно в базе данных, а также получать данные и формировать отчеты на основе полученных данных для анализа. В качестве примера можно привести интерактивные страницы банков, страницы для контроля товарных

запасов, социологические исследования и опросы, а также формы для обратной связи с пользователями.

Веб-приложение может использоваться для обновления веб-сайтов с периодически меняющимся содержанием.

### 1.3 Обзор аналогичных решений

Одним из ключевых моментов в разработке программного обеспечения является изучение аналогов, выявление достоинств и недостатков в них. С помощью анализа аналогов можно выделить функционал, который обязательно должен присутствовать в веб-приложении. Это необходимо для построения каркаса будущего приложения. Также анализ помогает выделить недостатки и избежать их в собственной реализации.

#### 1.3.1 Веб-приложение Duolingo.com

*Duolingo*[2] — самая популярная платформа для самостоятельного изучения языков. Данное веб-приложение имеет следующие возможности:

- изучение грамматических правил и слов по определенным темам в виде игрового процесса;
- тематические диалоги, которые позволяют приблизиться к жизненным ситуациям;
- личная статистика прогресса и статистика друзей;
- система достижений и бонусов за получение этих достижений.

Интерфейс веб-приложения представлен на рисунке 1.1.

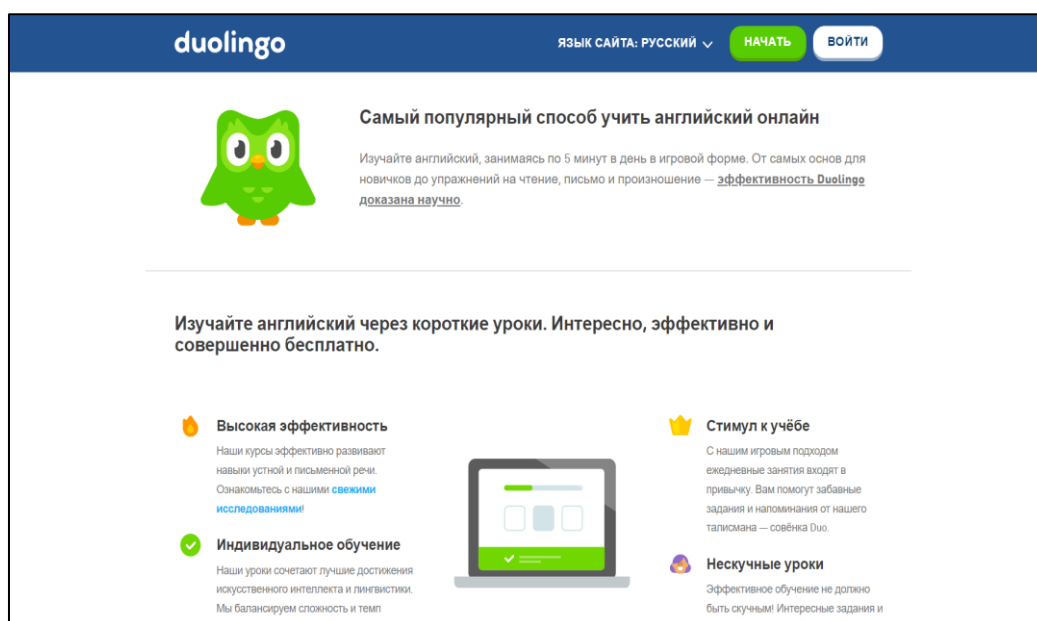


Рисунок 1.1 – Веб-приложение *duolingo.com*

Из недостатков необходимо отметить то, что имеющийся контент позволяет получить только начальные навыки овладения языком. Также нет возможности

начать обучение определённых тем или уроков, не пройдя до этого предыдущие. Также невозможно настроить приложение под свои навыки. У приложения есть и плюсы, например, за различные достижения пользователю вручаются награды (за количество выученных слов, за ежедневные посещения, за верные ответы подряд и так далее). Отслеживая статистику других пользователей, повышается мотивация изучения и прохождения уровней дальше.

### 1.3.2 Веб-приложение Poliglot16.ru

*Poliglot16*[3] — самоучитель английского языка с бесплатными уроками [2]. Приложение имеет следующие возможности:

- практические и тестовые задания по грамматике;
- изучение слов по тематическим словарям;
- добавление собственных словарей;
- сохранение данных приложения для переноса;
- настройка режимов обучения.

Интерфейс веб-приложения представлен на рисунке 1.2.

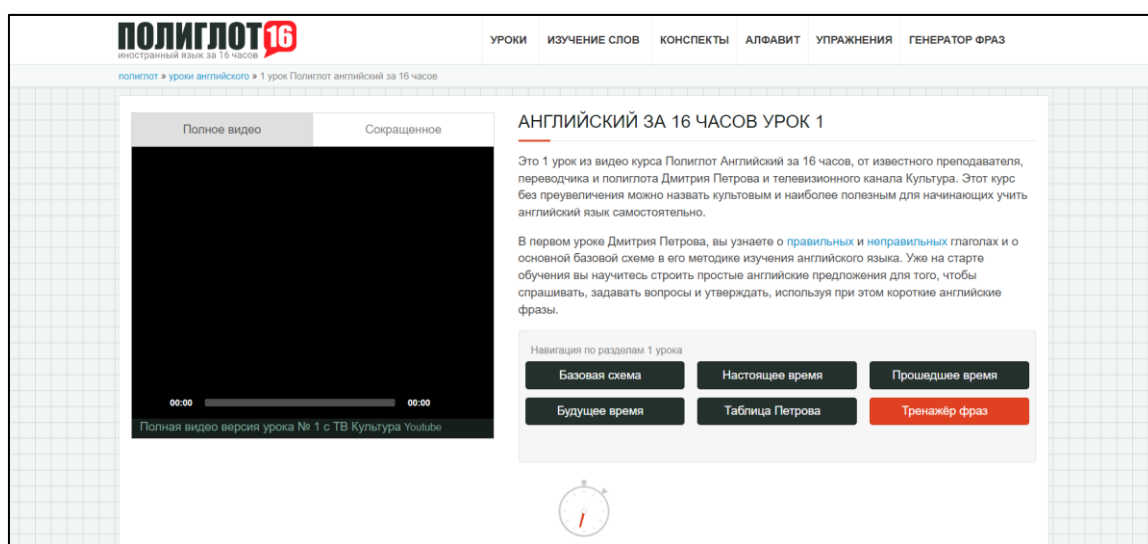


Рисунок 1.2 – Веб-приложение poliglot16.ru

Довольно устаревший дизайн и не обновляемая учебная база, что делает веб-приложение быстро выходящим из актуальности.

### 1.2.3 Веб-приложение Puzzel-english.com

Puzzel-english.com[4] – это приложение так же было разработано под мобильные операционные системы *IOS* и *Android* но имеет и веб-версию. Имеет следующие возможности:

- уроки по грамматическим правилам и расширение словарного запаса;
- конспект с грамматическими правилами;
- тематические истории в процессе прохождения, которых хорошо запоминаются слова за счёт образования ассоциаций;

- видео-уроки с текстовым сопровождением;
  - возможность установить напоминание о занятии на любое время.
- Интерфейс веб-приложения представлен на рисунке 1.3.

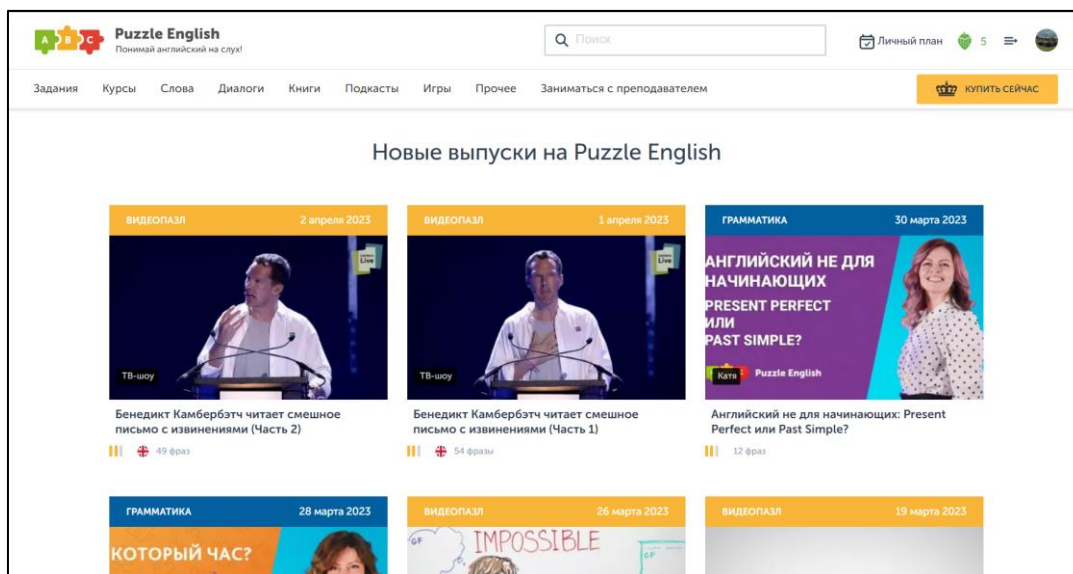


Рисунок 1.3 – Веб-приложение *puzzle-english.com*

К недостаткам данного веб-приложения можно отнести:

- основная часть контента является платной;
- невозможность изучать грамматику и лексику отдельно;
- довольно короткие уроки, за которые сложно усвоить тему.

## 1.4 Анализ прототипов

Схожесть рассмотренных программных средств заключается в том, что все они имеют как грамматические задания, так и задания на расширение лексикона. Также эти приложения содержат контент, который позволяет получить лишь начальные знания языка.

Главные отличия программ кроются в их функциональности. Так, например, в приложениях *Duolingo* и *Puzzle-English* изучение грамматики и словарного запаса проходит в модульной форме, в отличие от приложения *Poliglot16*. Только в приложении *Duolingo* имеется статистика прогресса, в приложении *Poliglot16* возможность добавлять собственные словари, а в приложении *Puzzle-English* есть грамматические конспекты. Также приложение *Puzzle-English* основную часть своего контента предлагает на платной основе.

Таким образом, можно сделать вывод о том, что в каждом приложении есть свои плюсы и минусы. Но, разумеется, каждому разработчику хочется, чтобы его приложение не имело отрицательных качеств, так как это сильно влияет рейтинг приложения, от чего напрямую зависит прибыль и статус самого разработчика.



## 2 Проектирование веб-приложения

### 2.1 Функциональные возможности веб-приложения

Приложение поддерживает 3 роли: «Гость», «Пользователь» и «Администратор». Диаграмма вариантов использования приложения указана на рисунке 2.1

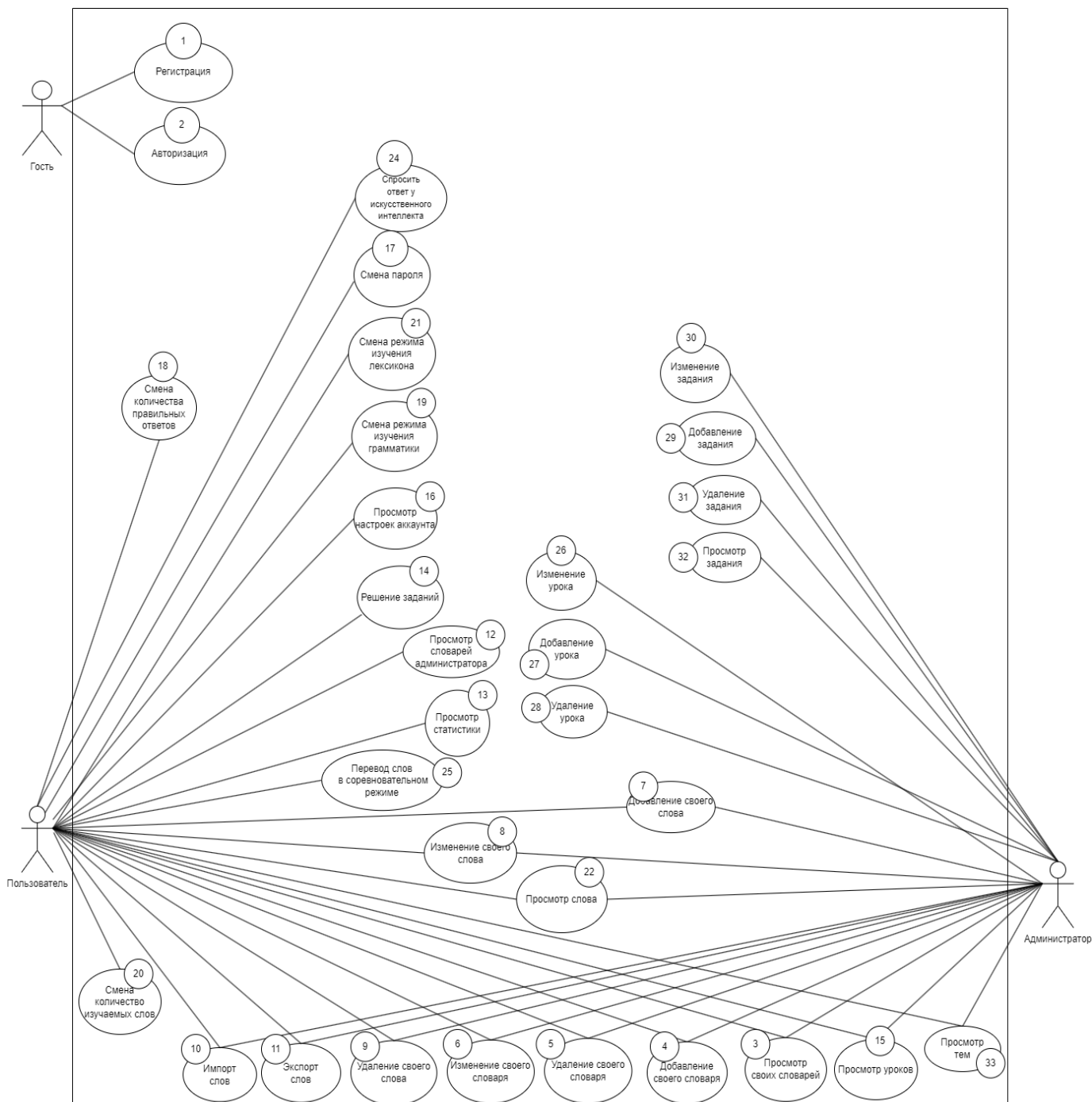


Рисунок 2.1 - Диаграмма вариантов использования

Диаграммы вариантов использования точно определяет функциональность веб-приложения, а также какие возможности будут доступны клиенту с определенной ролью.

Список ролей и их предназначение определены в таблице 2.1

Таблица 2.1 – назначение ролей пользователей

Роль	Назначение
Гость	Регистрация, авторизация
Пользователь	Просмотр своих словарей, просмотр словарей Администратора, добавление своего словаря, изменение своего словаря, удаление своего словаря, импорт своих слов, экспорт своих слов, просмотр слов своего словаря, удаление своих слов, изменение своих слов, импорт и экспорт слов, просмотр уроков, решение заданий урока, просмотр теоретических сведений урока, просмотр статистики, просмотр настроек аккаунта, смена пароля, смена режима изучения грамматики, смена количества изучаемых слов, смена режима изучения лексикона, Смена количества правильных ответов, тренировка грамматики.
Администратор	Просмотр своих словарей, добавление словаря, изменение словаря, удаление словаря, импорт и экспорт слов, просмотр слов словаря, удаление слов, изменение слов, просмотр уроков, добавление уроков, удаление уроков, изменение уроков, просмотр заданий урока, добавление заданий, изменение заданий, удаление заданий,

Функциональные возможности согласно диаграмме вариантов использования для ролей и их описание представлены в таблице 2.2

Таблица 2.2 – Функциональные возможности с описанием для каждой роли

Номер	Вариант использования	Роли	Описание
1	Регистрация	Гость	Возможность создать учетную запись для получения дополнительных возможностей.
2	Авторизация	Гость	Возможность входить в учетную запись.
3	Просмотр своих словарей	Пользователь, Администратор	Возможность просматривать словари.
4	Добавление своего словаря	Пользователь, Администратор	Возможность создавать словарь
5	Удаление своего словаря	Пользователь, Администратор	Возможность удалять словарь
6	Изменение своего словаря	Пользователь, Администратор	Возможность изменять свой словарь

Продолжение таблицы 2.2.

7	Добавление своего слова	Пользователь, Администратор	Возможность добавления слова в свой словарь
8	Изменение своего слова	Пользователь, Администратор	Возможность изменять слово в своем словаре
9	Удаление своего слова	Пользователь, Администратор	Возможность удалять слово из своего словаря
10	Импорт слов	Пользователь, Администратор	Возможность импортировать слова
11	Экспорт слов	Пользователь, Администратор	Возможность экспортировать слова
12	Просмотр словарей Администратора	Пользователь	Возможность просмотреть словари созданные Администратором
13	Просмотр статистики	Пользователь	Возможность просмотреть статистику по выученным словам, решенным заданиям, пройденных слов в квизе
14	Решение заданий	Пользователь	Возможность решать грамматические задания
15	Просмотр уроков	Пользователь, Администратор	Возможность просмотра всех доступных уроков
16	Просмотр настроек аккаунта	Пользователь	Возможность просмотра персональных настроек аккаунта
17	Смена пароля	Пользователь	Возможность сменить пароль
18	Смена количества правильных ответов	Пользователь	Возможность изменить количество правильных ответов, следующих подряд, после которого слово считается выученным и исключается из списка, но остаётся в словаре
19	Смена режима изучения грамматики	Пользователь	Возможность изменить способ изучения грамматики
20	Смена количества изучаемых слов	Пользователь	Возможность изменить количество слов изучаемых одновременно
21	Смена режима изучения лексикона	Пользователь	Возможность изменить способ изучения лексикона

Продолжение таблицы 2.2

Номер	Вариант использования	Роли	Описание
22	Просмотр слова	Пользователь, Администратор	Возможность просмотреть слова в словаре
24	Спросить ответ у искусственного интеллекта	Пользователь	Вовремя решения заданий можно спрашивать у искусственного интеллекта ответ с пояснением.
25	Перевод слов в соревновательном режиме	Пользователь	Возможность переводить случайные слова на скорость
26	Изменение урока	Администратор	Возможность добавлять новый урок
27	Добавление урока	Администратор	Возможность редактировать существующий урок
28	Удаление урока	Администратор	Возможность удалять урок
29	Добавление задания	Администратор	Возможность добавлять задание к уроку
30	Изменение задания	Администратор	Возможность изменять задание
31	Удаление задания	Администратор	Возможность удалять задание из урока
32	Просмотр заданий	Администратор	Возможность смотреть задания уроков

## 2.2 Логическая схема базы данных

Диаграмма базы данных таблиц (*Database Table Diagram*) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.2.

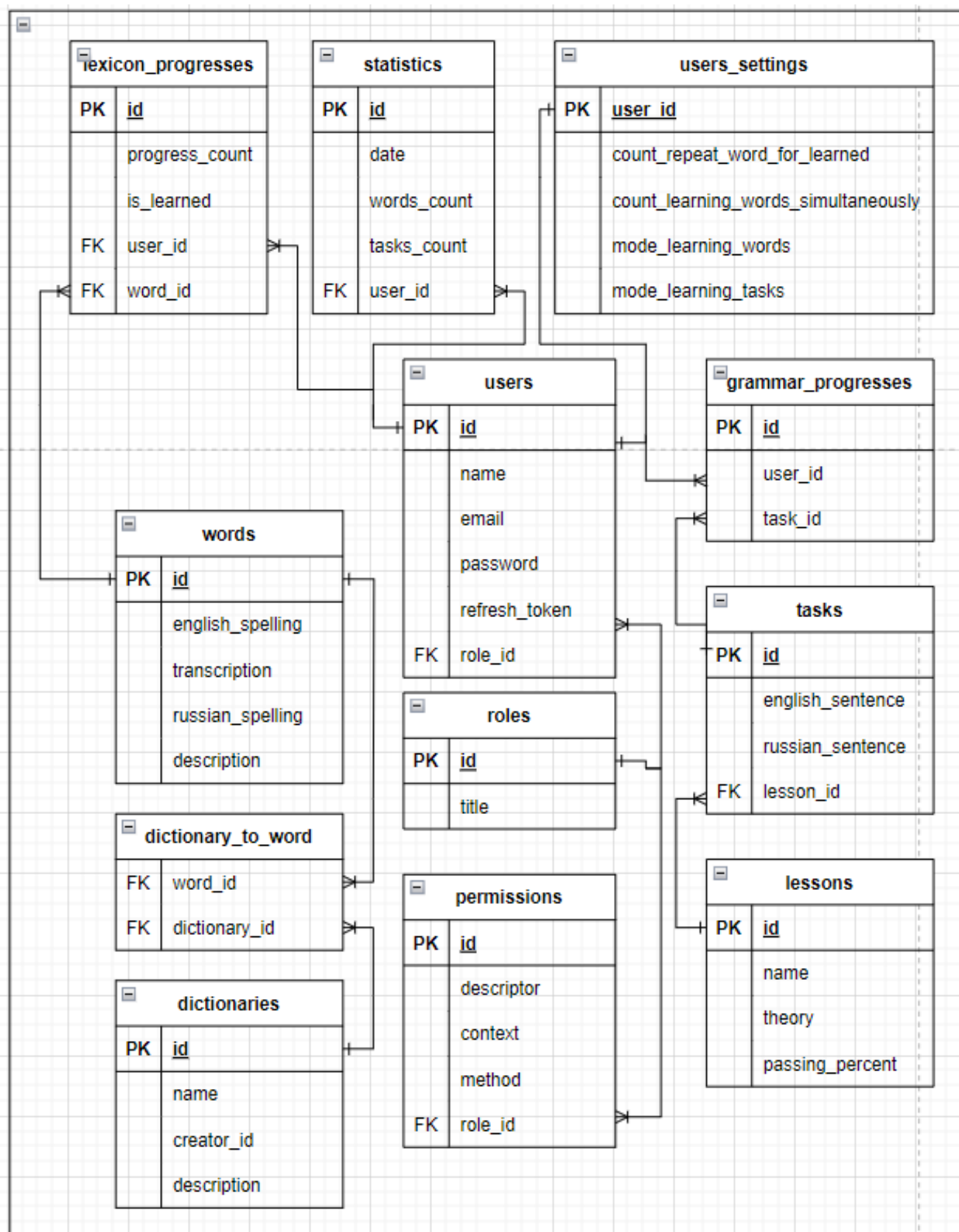


Рисунок 2.2 – Логическая схема базы данных

Таблица dictionaries содержит словари. Перечень полей таблицы dictionaries приведен в таблице 2.3.

Таблица 2.3 – Описание полей таблицы dictionaries

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор словаря. Является первичным ключом таблицы
name	varchar(30), not null	Название словаря
description	varchar(255)	Краткое описание словаря
creator_id	int, not null	Идентификатор создателя словаря

Таблица dictionary\_to\_word служит для связи таблиц dictionaries и words отношением многие ко многим. Перечень полей таблицы dictionary\_to\_word приведен в таблице 2.4.

Таблица 2.4 – Описание полей таблицы dictionary\_to\_word

Поле	Тип	Назначение
dictionary_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы dictionaries
word_id	int, not null, foreign key	внешний ключ, ссылающийся на идентификатор (id) таблицы words

Таблица grammar\_progresses служит для связи таблиц users и tasks отношением многие ко многим и каждая запись в таблице информирует о том, что определённый пользователь успешно выполнил определённое задание.

Перечень полей таблицы grammar\_progresses приведен в таблице 2.5.

Таблица 2.5 – Описание полей таблицы grammar\_progresses

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор связи, является первичным ключом таблицы
task_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы tasks
user_id	int not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы users

Таблица lessons хранит информацию о каждом уроке и имеет связь с таблицей tasks один ко многим, так как в одном уроке может быть несколько заданий.

Перечень полей таблицы lessons приведен в таблице 2.5.

Таблица 2.5 – Описание полей таблицы lessons

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор урока, является первичным ключом таблицы
name	varchar(30), not null	Название урока
theory	varchar(max), not null	Теоретические сведения по теме урока
passing_percent	int, not null	Минимальный процент правильно выполненных заданий урока, для того, чтобы тема урока считалась достаточно изученной

Таблица *lexicon\_progresses* служит для связи таблиц *users* и *words* отношением многие ко многим и каждая запись в таблице хранит информацию о прогрессе пользователя в изучении слов.

Перечень полей таблицы *lexicon\_progresses* приведен в таблице 2.6.

Таблица 2.7 – Описание полей таблицы *lexicon\_progresses*

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор связи, является первичным ключом таблицы
progress_count	int, not null	Количество правильных переводов слова пользователем, сделанных подряд
is_learned	Boolean, not null	Указывает, выучено слово ( <i>true</i> ) или нет ( <i>false</i> )
user_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы <i>users</i>
word_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы <i>words</i>

Таблица *permissions* хранит разрешения ролей для авторизации.

Перечень полей таблицы *permissions* приведен в таблице 2.7.

Таблица 2.7 – Описание полей таблицы *permissions*

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор разрешения, является первичным ключом таблицы
context	varchar(255)	Контекст выполняемого запроса
method	varchar(255), not null	Метод http запроса
role_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы <i>roles</i>
descriptor	int, not null	Наименование контроллера

Таблица *roles* хранит данные о каждой роли.

Перечень полей таблицы *roles* приведен в таблице 2.8.

Таблица 2.9 – Описание полей таблицы *roles*

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор роли, является первичным ключом таблицы
title	varchar(255), not null	Название роли

Таблица *statistics* хранит статистические данные об обучении пользователей за каждый день: количество правильных переводов слов, заданий и квиз очков.

Перечень полей таблицы *statistics* приведен в таблице 2.10.

Таблица 2.10 – Описание полей таблицы *statistics*

Поле	Тип	Назначение
id	int, identity, primary key	Идентификатор статистики, является первичным ключом таблицы
date	Date, not null	Дата создания записи

## Продолжение таблицы 2.10

words	int	Количество правильных переводов слова пользователем
tasks	int	Количество правильно сделанных заданий пользователем
quiz_points	int	Количество квиз очков, полученных пользователем за правильные переводы слов в квизе
user_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы roles

Таблица *tasks* хранит задания уроков.

Перечень полей таблицы *tasks* приведен в таблице 2.11.

Таблица 2.11 – Описание полей таблицы *tasks*

Поле	Тип	Назначение
id	int unique foreign key	Идентификатор задания, является первичным ключом таблицы
spanish_sentence	varchar(300) not null	Задание, написанное на испанском языке
russian_sentence	varchar(300) not null	Перевод задания из поля spanish_sentence на русский язык
lesson_id	int not null foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы lesson

Таблица *users* хранит данные пользователей.

Перечень полей таблицы *users* приведен в таблице 2.12.

Таблица 2.12 – Описание полей таблицы *users*

Поле	Тип	Назначение
id	int, unique, primary key	Идентификатор пользователя, является первичным ключом таблицы
name	varchar(255), not null	Имя пользователя
email	varchar(255), not null	Электронная почта пользователя
password	varchar(255), not null	Захэшированный пароль пользователя
role_id	int, not null, foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы roles
refresh_token	varchar(max), not null, foreign key	<i>refresh</i> -токен пользователя
id_enable_lesson	int, not null	Идентификатор (id) урока, доступного пользователю

Таблица *user\_settings* хранит настройки обучения пользователей.

Перечень полей таблицы *user\_settings* приведен в таблице 2.13.



Таблица 2.13 – Описание полей таблицы *user\_settings*

Поле	Тип	Назначение
user_id	int unique foreign key	Внешний ключ, ссылающийся на идентификатор (id) таблицы users
count_repeat_word_for_learned	int not null	Количество правильных ответов подряд, чтобы слово считалось выученным
count_learning_words_simultaneously	int not null	Количество слов изучаемы одновременно
mode_learning_words	varchar(255) not null	Способ изучения лексикона
mode_learning_tasks	varchar(255) not null	Способ изучения грамматики

Таблица *words* хранит слова, добавленные администратором и пользователями.

Перечень полей таблицы *words* приведен в таблице 2.14

Таблица 2.14 – Описание полей таблицы *words*

Поле	Тип	Назначение
id	int unique foreign key	Идентификатор слова, является первичным ключом таблицы
spanish_spelling	int not null	Испанское написание слова
transcription	int not null	Транскрипция слова на испанском
rus-sian_spelling	varchar(255) not null	Русский перевод слова из поля spanish_spelling
description	varchar(255) not null	Описание слова (для уточнения перевода)

Таблица *topics* хранит темы уроков.

Перечень полей таблицы *topics* приведен в таблице 2.15

Таблица 2.15 – Описание полей таблицы *words*

Поле	Тип	Назначение
id	int unique foreign key	Идентификатор темы, является первичным ключом таблицы
name	varchar(255) not null	Название темы для уроков

Таким образом, каждая таблица содержит определённые поля, связывающие одну таблицу с другой для эффективного управления данными.

## 2.3 Архитектура веб-приложения

Архитектура веб-приложения включает клиентскую часть для взаимодействия с пользователем, сервер для обработки запросов и базу данных для хранения информации. Эти компоненты обмениваются данными через сеть, обеспечивая работу системы.

Архитектура приложения детально представлена на рисунке 2.3

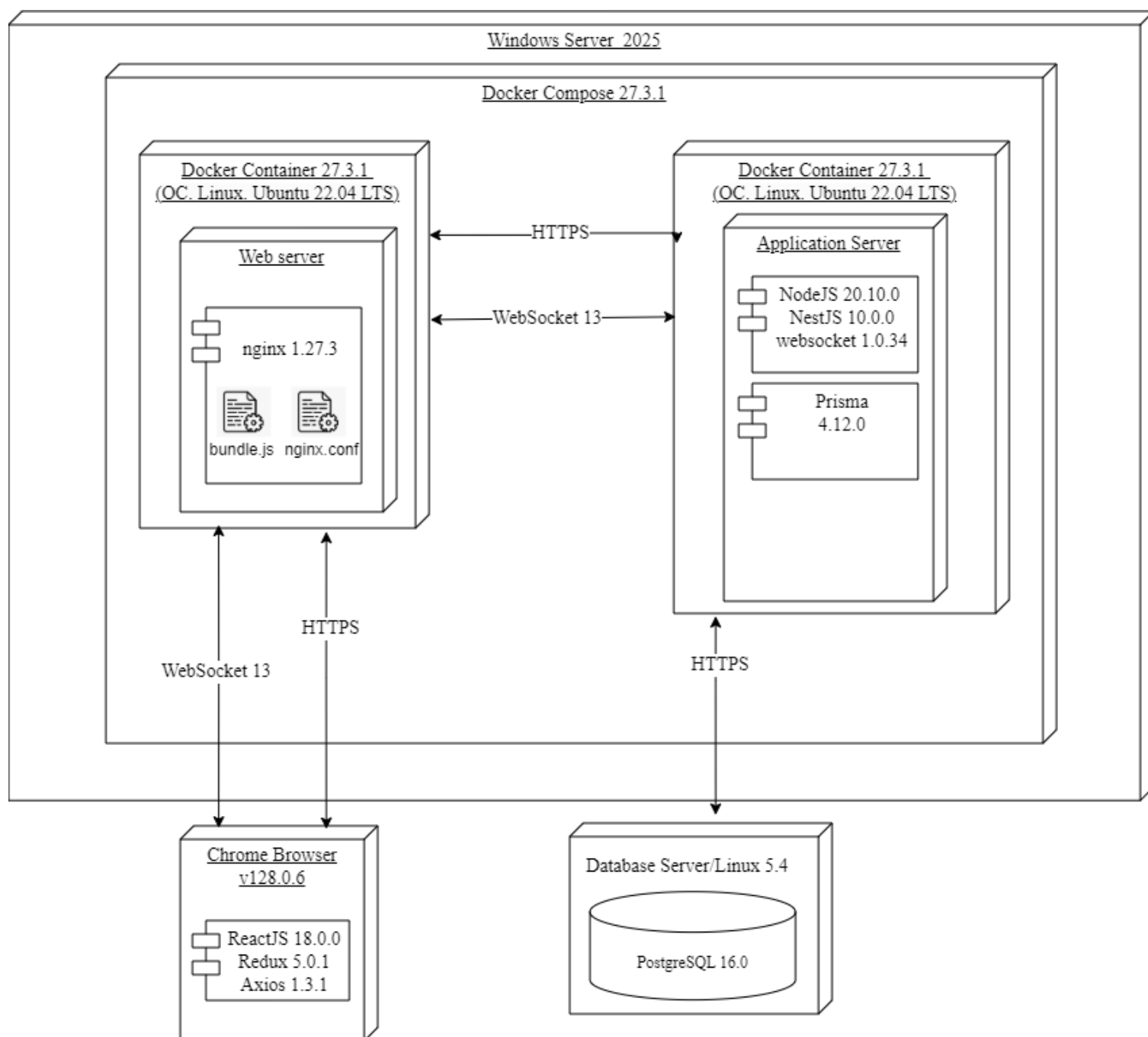


Рисунок 2.3 – Архитектура web-приложения

В таблице 2.16 подробно описаны основные элементы системы, их назначение и роль в работе приложения.

Таблица 2.16 – Назначение элементов архитектурной схемы веб-приложения

Элемент	Назначение
<i>Web Server (nginx[5])</i>	Предоставляет доступ к статическим ресурсам пользовательского интерфейса веб-приложения.
<i>Database Server (PostgreSQL[6])</i>	Используется для хранения, управления и предоставления доступа к данным, которые необходимы для работы веб-приложения.

Продолжение таблицы 2.16

<i>Application Server</i>	Обрабатывать запросы пользователя, запрашивать данные из базы данных.
---------------------------	---

Описание протоколов, используемых при работе веб-приложений, представлено в таблице 2.17.

Таблица 2.17 – Описание используемых протоколов.

Протокол	Назначение
HTTP[7]	Основной протокол для обмена данными.
HTTPS[8]	Обмен данными между Web Server и Application Server, Web Server и Chrome Browser. Обеспечить безопасную передачу данных путём использования криптографического протокола TLS. Используется транспортный протокол TCP[9]
WebSocket[10]	Обмен данными между Web Server и Application Server, Web Server и Chrome Browser. Обеспечивать постоянное соединения используя TCP-соединение. <i>Sec-WebSocketVersion 13</i>

Слаженная работа всех элементов архитектуры обеспечивает надёжную работу веб-приложения.

## 2.4 Вывод по разделу

1. Рассмотрены три роли web-приложения: гость, пользователь и администратор, двадцать две функциональные возможности с их описанием.
2. Определена структура базы данных, она включает четырнадцать таблиц. Описано содержание таблиц, их связи между друг другом.

## 3 Реализация веб-приложения

### 3.1 Программная платформа Node.js

Для серверной части проекта выбрали Node.js — высокопроизводительную платформу с событийной архитектурой, позволяющей обрабатывать множество запросов одновременно. Основным фреймворком стал Nest.js[11], который использует TypeScript[12] для статической проверки типов, упрощённой отладки и повышения читаемости кода. TypeScript обеспечивает совместимость с любыми средами JavaScript[13], включая Node.js, и поддерживает современные стандарты ECMAScript[14].

### 3.2 СУБД PostgreSQL

Для проекта выбрали PostgreSQL — мощную реляционную СУБД, известную надёжностью, расширяемостью и активным сообществом. Она обеспечивает хранение данных, поддержку ACID-транзакций и эффективную обработку сложных запросов, что делает её отличным выбором для систем с большими объёмами данных и сложной бизнес-логикой.

### 3.3 Prisma ORM

*Prisma*[15] — инструмент для работы с реляционными базами данных в проектах на TypeScript и JavaScript. Он реализует ORM нового поколения, упрощая взаимодействие с базой данных через Prisma Schema и минимизируя необходимость в сложных SQL-запросах. Интеграция с TypeScript обеспечивает автодополнение, проверку типов и безопасную работу с данными. Также Prisma поддерживает миграции схемы, упрощая управление структурой данных и командную работу.

Соответствие моделей Prisma и СУБД *PostgreSQL* представлено в таблице 3.1.

Таблица 3.1 – Сопоставление моделей, используемых в Prisma

Prisma название модели таблицы	<i>PostgreSQL</i> название таблицы
User	user
UserSettings	usersettings
Topic	topic
User	user
Lesson	lesson
Task	task
GrammarProgress	grammarprogress
Dictionary	dictionary
Word	word
DictionaryToWord	dictionarytoword

Продолжение таблицы 3.1

Prisma название модели таблицы	<i>PostgreSQL</i> название таблицы
--------------------------------	------------------------------------

LexiconProgress	lexiconprogress
Statistics	statistics
Session	session
Role	role
Permission	permission

Исходный код моделей на Prisma представлен в приложении А.

### 3.4 Библиотеки и фреймворки

В процессе разработки серверной части веб-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

Название	Версия	Описание
@nestjs[16]	9.0.0	Общие модули и утилиты фреймворка <i>NestJS</i> для построения приложений.
@prisma/client[17]	4.11.0	Генерируемый клиент для работы с базой данных, использующей <i>Prisma ORM</i> .
bcrypt[18]	5.1.0	Библиотека для хеширования паролей с использованием алгоритма <i>bcrypt</i> .
class-transformer[19]	0.5.1	Инструмент для преобразования объектов, например, при использовании DTO в приложении.
class-validator[20]	0.14.0	Библиотека для валидации данных на основе аннотаций в <i>TypeScript</i> -классах.
cookie[21]	0.5.0	Библиотека для обработки и работы с <i>HTTP</i> -куками.
cookie-parser[22]	1.4.6	<i>Middleware</i> для анализа <i>cookie</i> в <i>HTTP</i> -запросах.
cors[23]	2.8.5	<i>Middleware</i> для настройки <i>Cross-Origin Resource Sharing (CORS)</i> в приложении.
helmet[24]	7.0.0	<i>Middleware</i> для повышения безопасности <i>HTTP</i> -запросов в приложении.
https[25]	1.0.0	Модуль для создания <i>HTTPS</i> -серверов и работы с защищёнными соединениями.
joi[26]	17.9.0	Библиотека для валидации данных, удобная для настройки схем проверки.
nodemailer[27]	6.7.8	Библиотека для отправки электронных писем.

Продолжение таблицы 3.2

Название	Версия	Описание
<i>passport</i> [28]	0.6.0	Основная библиотека <i>Passport</i> для аутентификации.
<i>passport-jwt</i> [29]	4.0.1	<i>Passport</i> -стратегия для работы с <i>JWT</i> .
<i>reflect-metadata</i> [30]	0.1.13	Библиотека для работы с метаданными
<i>rxjs</i> [31]	7.8.1	Библиотека для работы с реактивными потоками данных.
<i>websocket</i> [32]	1.0.34	Библиотека для работы с <i>WebSocket</i> -соединениями.

В процессе разработки клиентской части веб-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

Название	Версия	Описание
<i>@fortawesome</i> [33]	6.3.0	библиотека <i>FontAwesome</i> для работы с <i>SVG</i> -иконками.
<i>@fortawesome/react-fontawesome</i> [34]	0.2.0	Компоненты для интеграции <i>FontAwesome</i> с <i>React</i> -приложениями.
<i>@testing-library/jest-dom</i> [35]	5.16.4	Библиотека для тестирования <i>React</i> -компонентов с использованием <i>Jest</i>
<i>@testing-library/react</i> [36]	13.5.0	Библиотека для эмуляции пользовательских событий в тестах.
<i>axios</i> [37]	1.3.3	HTTP-клиент для отправки запросов к серверу.
<i>bootstrap</i> [38]	5.2.3	CSS[39]-фреймворк для создания адаптивных и стилизованных пользовательских интерфейсов.
<i>chart.js</i> [39]	4.3.0	Библиотека для визуализации данных с использованием интерактивных графиков и диаграмм.
<i>date-fns</i> [40]	2.29.3	Библиотека для работы с датами в JavaScript.
<i>dotenv</i> [41]	16.0.3	Загрузка переменных окружения из файла <i>.env</i>
<i>js-cookie</i> [42]	3.0.1	Библиотека для работы с HTTP-куками в браузере.
<i>jwt-decode</i> [43]	3.1.2	Утилита для декодирования <i>JWT</i> -токенов.
<i>mobx</i> [44]	6.0.5	Библиотека для управления состоянием в приложениях.
<i>primeicons</i> [45]	6.0.1	Иконки для библиотеки компонентов <i>PrimeReact</i> .
<i>primereact</i> [46]	9.2.3	Библиотека компонентов пользовательского интерфейса для <i>React</i> .

Продолжение таблицы 3.3

<i>react</i> [47]	18.0.0	JavaScript-библиотека для создания пользовательских интерфейсов.
<i>react-bootstrap</i> [48]	2.7.4	Реализация компонентов Bootstrap для React.
<i>react-date-range</i> [49]	1.4.0	Компонент для выбора диапазона дат в React-приложениях.
<i>react-dom</i> [50]	18.2.0	Пакет для рендеринга React-компонентов в DOM.
<i>react-router-dom</i> [51]	6.10.0	Библиотека для работы с маршрутизацией в React-приложениях.
<i>react-scripts</i> [52]	5.0.1	Скрипты и конфигурация для работы с приложениями на Create React App.
<i>rxjs</i> [53]	7.8.1	Библиотека для работы с реактивными потоками данных.
<i>socket.io-client</i> [54]	4.6.1	Клиентская библиотека для работы с WebSocket через Socket.IO.
<i>web-vitals</i> [55]	2.1.4	Библиотека для измерения и анализа показателей производительности веб-приложений.
<i>ws</i> [56]	8.13.0	Библиотека для работы с WebSocket-соединениями.

### 3.5 Структура серверной части

Разработка серверной части веб-приложения выполнена с использованием модульно-слоистой архитектуры (Modular Layered Architecture) [57], которая обеспечивает чёткое разделение ответственности между компонентами системы. Такой подход упрощает масштабирование проекта, его поддержку и дальнейшее расширение. В архитектуре приложения выделяются следующие основные слои:

1. Контроллеры (Controllers): обрабатывают входящие HTTP-запросы, управляют входящими данными, вызывают бизнес-логику из сервисов и возвращают ответ клиенту.

2. Сервисы (Services): реализуют бизнес-логику приложения и обращаются к моделям для взаимодействия с базой данных.

3. Репозитории (Repository): инкапсулирует операции взаимодействия с базой данных. Модели данных, определённые с использованием Prisma, предоставляют абстракцию для доступа к таблицам.

4. Модули (Modules): они определяют зависимости, провайдеры и экспортируемые компоненты.

Файлы проекта организованы таким образом, чтобы они были сгруппированы в директории по их назначению. Так, каждая директория отвечает за определённый слой или функционал приложения. В таблице 3.4 приведён список директорий проекта разработки приложения и назначение файлов, хранящихся в этих директориях.

Таблица 3.4 – Директории проекта и их назначение

Директория	Назначение файлов директории
src	Главная рабочая директория проекта, содержит основную бизнес-логику приложения.
src/modules	Файлы модулей, отвечающие за обработку HTTP-запросов, взаимодействие с сервисами и возврат ответов клиенту
src/configuration	Содержит файлы конфигурации приложения
src/helpers	Хранит вспомогательные модули сторонних сервисов
prisma	Содержит файлы, связанные с ORM Prisma, служащие для описания схемы базы данных, файлы миграций

В каждой директории содержатся файлы, выполняющие строго определённые задачи. В таблице 3.5 приведен список файлов директории *modules* на примере *dictionary*.

Таблица 3.5 – список файлов директории *dictionary* в *modules*

Директория	Назначение файлов директории
dto	Здесь хранятся Data Transfer Objects, которые определяют структуру данных, передаваемых между клиентом и сервером
response	Папка с типами или интерфейсами для описания структуры ответов API, связанных со словарём
dictionary.controller.ts	Контроллер для обработки запросов, связанных со словарём (например, добавление, обновление или получение слов)
dictionary.module.ts	Файл, который связывает все элементы модуля. В NestJS он определяет зависимости, провайдеры и экспортируемые компоненты
dictionary.repository.ts	файл для работы с базой данных. Обычно отвечает за взаимодействие с таблицей или коллекцией, связанной со словарём
dictionary.service.ts	Содержит бизнес-логику для работы с данными словаря. Используется контроллером и может обращаться к repository

Пример реализации модуля предоставлен в Приложениях Г по Приложение Е. Всего разработано 11 модулей, каждый из которых методы, обрабатывающие входящие запросы. Для взаимодействия с частью методов необходима авторизация:

- *auth* для регистрации и авторизации пользователей;
- *dictionary* для работы со словарями;
- *grammar-progress* для работы со статистикой по заданиям;
- *lesson* для работы с уроками;
- *lexicon-progress* для работы со статистикой по словам;
- *session* для работы с *refresh*-токеном пользователей;
- *statistics* для работы со статистикой пользователей;
- *task* для работы с заданиями;
- *topic* для работы с темами уроков;



- *user* для смены пароля аккаунта и получения данных пользователей;
- *user-settings* для работы с настройками аккаунта пользователя;
- *word* для работы со словами.

Всего в проекте 33 *API* метода. Описание доступных модулей, адресов и методов приведено в таблице 3.6.

Таблица 3.6 – Описание методов *API*

Адрес	Метод	Описание	Модуль
<i>/api/user/{id}</i>	<i>GET</i>	Получение данных пользователя по его <i>id</i>	<i>user</i>
<i>/api/user</i>	<i>PATCH</i>	Обновление пароля пользователя	<i>user</i>
<i>/api/auth/registration</i>	<i>POST</i>	Регистрация нового пользователя	<i>auth</i>
<i>/api/auth/login</i>	<i>POST</i>	Вход в аккаунт	<i>auth</i>
<i>/api/auth/refresh</i>	<i>POST</i>	Обновление токенов по <i>refresh</i> -токену	<i>auth</i>
<i>/api/auth/logout</i>	<i>POST</i>	Выход из аккаунта	<i>auth</i>
<i>/api/dictionary/</i>	<i>POST</i>	Создание словаря	<i>dictionary</i>
<i>/api/dictionary/admin</i>	<i>GET</i>	Получение списка словарей, созданных администратором	<i>dictionary</i>
<i>/api/dictionary/user</i>	<i>GET</i>	Получение списка словарей, созданных пользователем	<i>dictionary</i>
<i>/api/dictionary/review</i>	<i>GET</i>	Получение списка словарей, созданных пользователем и администратором	<i>dictionary</i>
<i>/api/dictionary/learn</i>	<i>GET</i>	Получение списка словарей для обучения	<i>dictionary</i>
<i>/api/dictionary/{id}</i>	<i>PATCH</i>	Обновление словаря по <i>id</i>	<i>dictionary</i>
<i>/api/dictionary/{id}</i>	<i>DELETE</i>	Удаление словаря по <i>id</i>	<i>dictionary</i>
<i>/api/word/{id}</i>	<i>POST</i>	Создание слова	<i>word</i>
<i>/api/word/import/{id}</i>	<i>POST</i>	Импорт слов в словарь по <i>id</i> словаря	<i>word</i>
<i>/api/word/{id}</i>	<i>PATCH</i>	Обновление слова по <i>id</i>	<i>word</i>
<i>/api/word/{id}/dictionary/{dictionaryId}</i>	<i>DELETE</i>	Удаление слова из словаря по <i>id</i> слова и <i>id</i> словаря	<i>word</i>
<i>/api/lesson</i>	<i>POST</i>	Создание урока	<i>lesson</i>
<i>/api/lesson/admin</i>	<i>GET</i>	Получение всех уроков	<i>lesson</i>
<i>/api/lesson/learn</i>	<i>GET</i>	Получение доступных уроков для изучения пользователем	<i>lesson</i>
<i>/api/lesson/{id}</i>	<i>PATCH</i>	Обновление урока по <i>id</i>	<i>lesson</i>
<i>/api/lesson/{id}</i>	<i>DELETE</i>	Удаление урока по <i>id</i>	<i>lesson</i>
<i>/api/grammar-progress</i>	<i>POST</i>	Создание записи изученного урока	<i>grammar-progress</i>
<i>/api/task</i>	<i>POST</i>	Создать задания	<i>task</i>
<i>/api/task/{id}</i>	<i>PATCH</i>	Обновление задания по <i>id</i>	<i>task</i>
<i>/api/task/{id}</i>	<i>DELETE</i>	Удаление задания по <i>id</i>	<i>task</i>
<i>/api/statistics</i>	<i>POST</i>	Создание статистики	<i>statistics</i>
<i>/api/statistics/all</i>	<i>GET</i>	Получение всех статистик	<i>statistics</i>
<i>/api/statistics/user-all</i>	<i>GET</i>	Получение всех статистик пользователя	<i>statistics</i>
<i>/api/statistics/user-to-day</i>	<i>GET</i>	Получение статистики пользователя за текущий день	<i>statistics</i>
<i>/api/user-settings</i>	<i>GET</i>	Получение настроек пользователя	<i>user-settings</i>
<i>/api/user-settings</i>	<i>PATCH</i>	Обновление настроек пользователя	<i>user-settings</i>
<i>/api/lexicon-progress</i>	<i>POST</i>	Создание записи изученного слова	<i>lexicon-progress</i>

### 3.6 Сторонние сервисы

Для успешной реализации функционала веб-приложения были использованы следующие сторонние сервисы:

1. SMTP[58] — интеграция с этим сервисом позволила реализовать отправку электронных писем из приложения.
2. DeepInfra AI[59] — используется для интеграции искусственного интеллекта в приложение, включая обработку естественного языка (NLP), генерацию текста, анализ данных и другие задачи, связанные с машинным обучением. Это позволяет улучшить функциональность приложения, добавив возможности автоматизации, персонализации и интеллектуальной обработки информации.

### 3.7 Реализация функций для роли «Гость»

#### 3.7.1 Регистрация

Метод регистрации реализован в методе `registration` контроллера `AuthController`. Реализация метода представлена на листинге 3.1

```
@ApiResponse()
@HttpCode(HttpStatus.CREATED)
@Post('registration')
public registration(
    @Body() registrationDto: UserRegistrationDto,
): Promise<void> {
    return this.authService.registration(registrationDto);
}
```

Листинг 3.1 – Реализация метода `registration`

Метод `registration` принимает HTTP-запрос с телом, соответствующим структуре `UserRegistrationDto`. Этот объект содержит данные пользователя, необходимые для регистрации. Данные из `UserRegistrationDto` передаются в сервис `AuthService` для выполнения логики регистрации. Сервис выполняет проверку данных, их сохранение в базу данных и другие необходимые шаги для завершения регистрации пользователя. В случае успешного выполнения регистрации метод возвращает HTTP-статус 201 Created.

#### 3.7.2 Авторизация

Метод входа реализован в методе `login` контроллера `AuthController`. Реализация метода представлена на листинге 3.2

```
@ApiResponse(swaggerType(UserLoginResponse))
@HttpCode(HttpStatus.OK)
```

```

@UseGuards (LocalAuthGuard)
@Post ('login')
public login (@Req () req: RequestWithUser):
Promise<UserLoginResponse> {
    return this.authService.login (req);
}

```

Листинг 3.2 – Реализация метода login

Метод login принимает HTTP-запрос с учетными данными пользователя (например, логин и пароль). Для защиты метода используется LocalAuthGuard, который автоматически проверяет переданные учетные данные. Если данные не соответствуют требованиям, запрос отклоняется с ошибкой. В случае успешной аутентификации, метод вызывает функцию login сервиса AuthService, передавая данные запроса. Сервис генерирует пару токенов — access и refresh — для дальнейшей работы пользователя в системе. Сгенерированный access-токен, необходимый для доступа к защищенным ресурсам. refresh-токен, который используется для обновления access-токена.

### 3.8 Реализация функций для роли «Пользователь»

#### 3.8.1 Просмотр своих словарей

Метод просмотра своего словаря реализован в методе getUserDictionaries контроллера DictionaryController. Реализация метода представлена на листинге 3.3.

```

@ApiBearerAuth ()
@ApiOkResponse (swaggerType (DictionaryResponse))
@UseGuards (JwtAuthGuard)
@Get ('user')
public getUserDictionaries (
    @Req () req: RequestWithUser,
): Promise<DictionaryResponse []> {
    return this.dictionaryService.getUserDictionaries (+req.user.id);
}

```

Листинг 3.3 – Реализация метода getUserDictionaries

Метод getUserDictionaries позволяет пользователю получить список своих словарей. JwtAuthGuard, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой. После успешной аутентификации метод вызывает функцию getUserDictionaries сервиса DictionaryService, передавая идентификатор текущего пользователя (req.user.id). Метод возвращает массив объектов типа DictionaryResponse, каждый из которых содержит информацию о словаре: его название, описание, количество слов и другие метаданные.

### 3.8.2 Добавление словаря

Метод добавления словаря реализован в методе `createDictionary` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.4.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post()
public createDictionary(
    @Req() req: RequestWithUser,
    @Body() createDictionaryDto: CreateDictionaryDto,
): Promise<DictionaryResponse> {
    return this.dictionaryService.createDictionary(
        +req.user.id,
        createDictionaryDto,
    );
}
```

Листинг 3.4 – Реализация метода `createDictionary`

Метод принимает данные нового словаря (`CreateDictionaryDto`). Проверяет аутентификацию пользователя с помощью `JwtAuthGuard`. Вызывает функцию `createDictionary` из `DictionaryService`, передавая идентификатор пользователя и данные словаря. Возвращает объект типа `DictionaryResponse` с информацией о созданном словаре.

### 3.8.3 Удаление словаря

Метод просмотра своего словаря реализован в методе `remove` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.5.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('/:id')
public remove(
    @Param('id') id: string,
    @Req() req: RequestWithUser,
): Promise<void> {
    return this.dictionaryService.removeDictionary(+id, +req.user.id);
}
```

Листинг 3.5 – Реализация метода `remove`

Он принимает идентификатор словаря через параметр `id` и использует данные пользователя из запроса `req` для проверки прав доступа. Метод вызывает функцию `removeDictionary` из `DictionaryService`, чтобы удалить словарь, принадлежащий текущему пользователю. Если операция выполняется успешно, метод возвращает пустой ответ с HTTP-статусом 200 ОК. В случае ошибки возвращается соответствующий HTTP-статус.

### 3.8.4 Изменение своего словаря

Метод изменения словаря реализован в методе `updateDictionary` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.6.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(DictionaryResponse))
@UseGuards(JwtAuthGuard)
@Patch('/:id')
public updateDictionary(
  @Param('id') id: string,
  @Req() req: RequestWithUser,
  @Body() updateDictionaryDto: UpdateDictionaryDto,
): Promise<DictionaryResponse> {
  return this.dictionaryService.updateDictionary(
    +id,
    +req.user.id,
    updateDictionaryDto,
  );
}
```

Листинг 3.6 – Реализация метода `updateDictionary`

Метод обновляет словарь по указанному идентификатору. Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard` и вызывает функцию `updateDictionary` из `DictionaryService`, передавая идентификатор словаря, идентификатор пользователя и новые данные для обновления. В случае успеха метод возвращает обновленный объект типа `DictionaryResponse`.

### 3.8.5 Добавление своего слова

Метод добавления слова реализован в методе `createWord` контроллера `WordController`. Реализация метода представлена на листинге 3.7.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post('/:id')
public createWord(
  @Req() req: RequestWithUser,
  @Param('id') dictionaryId: string,
  @Body() createWordDto: CreateWordDto,
): Promise<WordForDictionaryResponse> {
  return this.wordService.createWord(
    +req.user.id,
    +dictionaryId,
    createWordDto,
  );
}
```

Листинг 3.7 – Реализация метода `createWord`

Метод добавляет новое слово в словарь, указанный через параметр `id`. Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard`, затем вызывает функцию `createWord` из `WordService`, передавая идентификатор пользователя, идентификатор словаря и данные нового слова. После успешного выполнения метод возвращает объект типа `WordForDictionaryResponse`, содержащий информацию о добавленном слове.

### 3.8.6 Изменение своего слова

Метод изменения слова реализован в методе `updateWord` контроллера `WordController`. Реализация метода представлена на листинге 3.8.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(WordResponse))
@UseGuards(JwtAuthGuard)
@Patch('/:id')
public updateWord(
    @Req() req: RequestWithUser,
    @Param('id') wordId: string,
    @Body() updateWordDto: UpdateWordDto,
): Promise<WordResponse> {
    return this.wordService.updateWord(
        +req.user.roleId,
        +wordId,
        updateWordDto,
    );
}
```

Листинг 3.8 – Реализация метода `updateWord`

Метод обновляет слово по указанному идентификатору `id`. Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard`, затем вызывает функцию `updateWord` из `WordService`, передавая идентификатор пользователя, идентификатор слова и новые данные для обновления. После успешного выполнения метод возвращает обновленный объект типа `WordResponse`, содержащий информацию о измененном слове.

### 3.8.7 Удаление своего слова

Метод удаления слова реализован в методе `removeWordFromDictionary` контроллера `WordController`. Реализация метода представлена на листинге 3.9.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('/:id/dictionary/:dictionaryId')
public removeWordFromDictionary(
    @Req() req: RequestWithUser,
    @Param('id') id: string,
    @Param('dictionaryId') dictionaryId: string,
): Promise<void> {
```

```

return this.wordService.removeWordFromDictionary(
    +req.user.id,
    +id,
    +dictionaryId,
);
}

```

Листинг 3.9 – Реализация метода `removeWordFromDictionary`

Метод удаляет слово из словаря по указанным идентификаторам `id` (слово) и `dictionaryId` (словарь). Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard`, затем вызывает функцию `removeWordFromDictionary` из `WordService`, передавая идентификаторы пользователя, слова и словаря. После успешного выполнения метод возвращает пустой ответ.

### 3.8.8 Импорт слов

Метод импорта слов реализован в методе `importDictionary` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.10.

```

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post('import/:id')
public async importDictionary(
    @Param('id') id: string,
    @Req() req: RequestWithUser,
    @Body() body: { property: any[] }, // Указываем ожидаемую
структуру
): Promise<void> {
    const wordData = body.property; // Извлекаем массив из property
    await this.dictionaryService.importDictionary(+id, wordData);
}

```

Листинг 3.10 – Реализация метода `importDictionary`

Метод `importDictionary` предназначен для импорта слов в словарь по указанному идентификатору `id` (словарь). Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard`, затем принимает данные через тело запроса (`@Body()`), где ожидается объект с массивом `property`, содержащим слова для импорта. Из тела запроса извлекается массив `property`, который передается в функцию `importDictionary` из `DictionaryService` вместе с идентификатором словаря (`id`). После успешного выполнения метод возвращает пустой ответ, указывая на завершение операции импорта.

### 3.8.9 Экспорт слов

Метод экспорта словаря реализован в методе `export` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.11.

```

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get('export/:id')
public async export(@Param('id') id: string, @Res() res:
ExpressResponse) {
    const fileName = 'dictionary_' + Date.now() + '.json';
    const fileContent = await
this.dictionaryService.exportDictionary(+id);

    fs.writeFileSync(fileName, JSON.stringify(fileContent), 'utf-8');

    res.setHeader('Content-Type', 'application/json');
    res.setHeader('Content-Disposition', `attachment;
filename=${fileName}`);

    const fileStream = fs.createReadStream(fileName);
    fileStream.pipe(res);

    fileStream.on('end', () => {
        fs.unlinkSync(fileName);
    });
}

```

Листинг 3.11 – Реализация метода export

Метод export предназначен для экспорта словаря по указанному идентификатору id (словарь). Он проверяет аутентификацию пользователя с помощью JwtAuthGuard, затем вызывает функцию exportDictionary из DictionaryService, передавая идентификатор словаря (id). Полученные данные словаря сохраняются в файл формата JSON с уникальным именем, сгенерированным на основе текущей временной метки. Метод устанавливает заголовки HTTP-ответа для скачивания файла и передает его пользователю через поток чтения файла. После завершения отправки файла временный файл удаляется. После успешного выполнения метод возвращает файл JSON, содержащий данные словаря, для скачивания пользователем.

### 3.8.10 Просмотр словарей администратора

Метод получения словарей администратора реализован в методе getAdminDictionaries контроллера DictionaryController. Реализация метода представлена на листинге 3.12.

```

@ApiBearerAuth()
@ApiOkResponse(swaggerType(DictionaryResponse))
@UseGuards(JwtAuthGuard)
@Get('admin')
public getAdminDictionaries(): Promise<DictionaryResponse[]> {
    return this.dictionaryService.getAdminDictionaries();
}

```

Листинг 3.12 – Реализация метода getAdminDictionary



Метод позволяет администратору получить список всех словарей. Он использует защиту `JwtAuthGuard` для проверки прав доступа и вызывает функцию `getAdminDictionaries` из `DictionaryService`, чтобы получить словари. В ответ возвращается массив объектов типа `DictionaryResponse`, содержащий информацию о словарях.

### 3.8.11 Просмотр статистики

Метод получения статистики пользователя за сегодня реализован в методе `getUserTodayStatistics` контроллера `StatisticsController`. Реализация метода представлена на листинге 3.15.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get('user-today')
private getUserTodayStatistics(
    @Req() req: RequestWithUser,
): Promise<StatisticsResponse> {
    return
    this.statisticsService.getUserTodayStatistics(+req.user.id);
}
```

Листинг 3.13 – Реализация метода `getUserTodayStatistic`

Метод позволяет пользователю получить статистику за текущий день. Он использует защиту `JwtAuthGuard` для проверки аутентификации и прав доступа, а затем вызывает функцию `getUserTodayStatistics` из `StatisticsService`, передавая идентификатор пользователя. Метод возвращает объект типа `StatisticsResponse`, который содержит данные статистики за сегодня для конкретного пользователя.

### 3.8.12 Решение заданий

Метод решения заданий реализован в методе `create` контроллера `GrammarProgressController`. Реализация метода представлена в листинге 3.14.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post()
create(
    @Req() req: RequestWithUser,
    @Body() createGrammarProgressDto: CreateGrammarProgressDto,
): Promise<void> {
    return this.grammarProgressService.createOrUpdateGrammarProgress(
        +req.user.id,
        createGrammarProgressDto.taskId,
    );
}
```

Листинг 3.14 – Реализация метода `create`

Метод `create` позволяет пользователю сохранить или обновить прогресс выполнения заданий по грамматике. Данный метод защищен с помощью `JwtAuthGuard`, который гарантирует, что запрос отправлен аутентифицированным пользователем. В случае отсутствия аутентификации запрос будет отклонен с соответствующей ошибкой. После успешной аутентификации метод принимает объект `CreateGrammarProgressDto` из тела запроса, который содержит идентификатор выполненного задания (`taskId`). Эти данные вместе с идентификатором пользователя передаются в сервис `GrammarProgressService`, где вызывается функция `createOrUpdateGrammarProgress`.

### 3.8.13 Просмотр уроков

Метод просмотра уроков реализован в методе `getLearnLessons` контроллера `LessonController`. Реализация метода представлена на листинге 3.15.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(LessonResponse))
@UseGuards(JwtAuthGuard)
@Get('learn')
public getLearnLessons(
    @Req() req: RequestWithUser,
): Promise<LessonResponse[]> {
    return this.lessonService.getLearnLessons(req);
}
```

Листинг 3.15 – Реализация метода `getLearnLessons`

Метод `getLearnLessons` предназначен для получения списка уроков, доступных для изучения. Он проверяет аутентификацию пользователя с помощью `JwtAuthGuard`, затем вызывает функцию `getLearnLessons` из `LessonService`, передавая объект запроса с информацией о пользователе (`req`). После успешного выполнения метод возвращает массив объектов `LessonResponse`, содержащих информацию об уроках, доступных для изучения.

### 3.8.14 Просмотр настроек аккаунта

Метод для просмотра настроек аккаунта реализован в контроллере `UserSettingsController`. В частности, метод `getUserSettings`, представленный в листинге 3.16, позволяет пользователю получить свои настройки.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get()
private getUserSettings(@Req() req: RequestWithUser):
    Promise<UserSettingsResponse> {
    return this.userSettingsService.getUserSettings(+req.user.id);
}
```

Листинг 3.16 – Реализация метода `getUserSettings`

Метод использует защиту `JwtAuthGuard` для проверки аутентификации пользователя и возвращает настройки пользователя в виде объекта `UserSettingsResponse`. Для получения данных вызывается метод `userSettingsService.getUserSettings`, который получает настройки пользователя по его ID, извлеченному из объекта запроса `req.user.id`.

### 3.8.15 Смена пароля

Метод смены пароля реализован в методе `changePassword` контроллера `UserController`. Реализация метода представлена на листинге 3.18.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@ApiOkResponse(swaggerType(UserResponse))
@Patch('password')
public changePassword(
    @Req() req: RequestWithUser,
    @Body() updateUserPasswordDto: UpdateUserPasswordDto,
): Promise<void> {
    return this.userService.changeUserPassword(req.user,
        updateUserPasswordDto);
}
```

Листинг 3.17 – Реализация метода `changePassword`

Метод `changePassword` позволяет пользователю сменить свой пароль. Он защищен с помощью `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой. После успешной аутентификации метод принимает данные о старом и новом пароле через объект `UpdateUserPasswordDto` из тела запроса и передает их в сервис `UserService`. Сервис `changeUserPassword` проверяет правильность старого пароля и обновляет его на новый, если все данные корректны. В случае ошибки, выбрасывается исключение.

### 3.8.16 Смена количества правильных ответов

Метод смены количества правильных ответов реализован в компоненте `PersonalCabinet`, который управляет настройками пользователя, включая настройки для обучения. Реализация метода представлена на листинге 3.18.

```
const changeSettingsHandler = (id, value) => {
    if (learningSettings[id] !== value) {
        setLearningSettings((prev) => ({ ...prev, [id]: value }));
    }
};

const updateUserSettingsRequest = async () => {
    try {
        await updateUserSettings(learningSettings);
    }
}
```

```

        userSettings.setUserSettings(learningSettings);
        showSuccess("Настройки обучения обновлены.");
    } catch (error) {
        showError();
    }
}; }

```

Листинг 3.18 – Реализация метода updateUserSettings

Метод изменения количества правильных ответов, которые идут подряд до того, как слово будет считаться выученным, реализуется с помощью компонента Slider. Когда пользователь изменяет значение слайдера, вызывается метод changeSettingsHandler, который обновляет состояние learningSettings. Эти данные затем отправляются на сервер через API запрос updateUserSettingsRequest, где обновляются настройки пользователя.

### 3.8.17 Смена режима изучения грамматики

Метод смены режима изучения грамматики также реализован в компоненте PersonalCabinet, где пользователь может выбрать способ изучения грамматики из предложенных вариантов. Смена режима осуществляется через компонент Dropdown, который обновляет состояние learningSettings при изменении значения. Реализация метода представлена на листинге 3.19.

```

const changeSettingsHandler = (id, value) => {
    if (learningSettings[id] !== value) {
        setLearningSettings((prev) => ({ ...prev, [id]: value }));
    }
};

const learningModes = [
    {
        name: "Перевод с испанского",
        value: LearningMode.TRANSLATE_FROM_ENGLISH,
    },
    { name: "Перевод с русского", value:
LearningMode.TRANSLATE_FROM_RUSSIAN },
    { name: "Комбинированный вариант", value: LearningMode.COMBINED },
];

```

Листинг 3.19 – Реализация метода updateLearningModeTasks

Метод смены режима изучения грамматики реализуется через элемент интерфейса Dropdown, который позволяет пользователю выбрать вариант режима. После того как пользователь выбирает новый режим, вызывается метод changeSettingsHandler, который обновляет соответствующее значение в состоянии learningSettings

### 3.8.18 Смена количества изучаемых слов

Метод смены количества изучаемых слов реализован в компоненте `PersonalCabinet`, который позволяет пользователю настроить количество слов, которые он хочет изучать одновременно, а также количество правильных ответов, после которого слово считается выученным. Для изменения этих настроек используются компоненты `Slider` для выбора числа слов и правильных ответов. Реализация метода представлена на листинге 3.20.

```
const changeSettingsHandler = (id, value) => {
  if (learningSettings[id] !== value) {
    setLearningSettings((prev) => ({ ...prev, [id]: value }));
  }
};

const [learningSettings, setLearningSettings] = useState({
  countRepeatWordForLearned:
userSettings.getCountRepeatWordForLearned(),
  countRepeatWordsSimultaneously:
userSettings.getCountRepeatWordsSimultaneously(),
  learningModeWords: userSettings.getLearningModeWords(),
  learningModeTasks: userSettings.getLearningModeTasks(),
});

<Slider
  className="learning-settings__input"
  value={learningSettings.countRepeatWordsSimultaneously}
  min={10}
  max={50}
  onChange={(e) =>
    changeSettingsHandler("countRepeatWordsSimultaneously", e.value)
  }
/>
<label>{learningSettings.countRepeatWordsSimultaneously}</label>

<Slider
  className="learning-settings__input"
  value={learningSettings.countRepeatWordForLearned}
  min={2}
  max={10}
  onChange={(e) =>
    changeSettingsHandler("countRepeatWordForLearned", e.value)
  }
/>
<label>{learningSettings.countRepeatWordForLearned}</label> }
```

Листинг 3.20 – Реализация метода `updateLearningModeWords`

Метод смены режима изучения лексикона реализован с использованием компонента `'Dropdown'`, который позволяет пользователю выбрать предпочтительный способ изучения слов. В данном случае предлагается несколько режимов, таких как

перевод с испанского, с русского или комбинированный вариант. Когда пользователь выбирает один из этих режимов, происходит обновление состояния приложения, и выбранный режим сохраняется в настройках пользователя. Это изменение в режиме обучения позволяет персонализировать процесс, что повышает его эффективность и делает его более удобным для каждого пользователя. После изменения режима, данные обновляются и сохраняются через соответствующий API-запрос, а пользователю отображается уведомление об успешном изменении настроек.

### 3.8.19 Смена режима изучения лексикона

Метод смены режима изучения лексикона реализован в компоненте PersonalCabinet, который позволяет пользователю выбрать способ изучения слов. В данном случае используется компонент Dropdown для выбора режима изучения лексикона. Реализация метода представлена в листинге 3.21.

```
const learningModes = [
  {
    name: "Перевод с испанского",
    value: LearningMode.TRANSLATE_FROM_ENGLISH,
  },
  { name: "Перевод с русского", value:
LearningMode.TRANSLATE_FROM_RUSSIAN },
  { name: "Комбинированный вариант", value: LearningMode.COMBINED },
];

const changeSettingsHandler = (id, value) => {
  if (learningSettings[id] !== value) {
    setLearningSettings((prev) => ({ ...prev, [id]: value }));
  }
};

const [learningSettings, setLearningSettings] = useState({
  countRepeatWordForLearned:
userSettings.getCountRepeatWordForLearned(),
  countRepeatWordsSimultaneously:
userSettings.getCountRepeatWordsSimultaneously(),
  learningModeWords: userSettings.getLearningModeWords(),
  learningModeTasks: userSettings.getLearningModeTasks(),
});

<Dropdown
  value={learningSettings.learningModeWords}
  onChange={(e) => changeSettingsHandler("learningModeWords",
e.value)}
  options={learningModes}
  optionLabel="name"
  placeholder="Выберите вариант"
  className="w-full"
/>
```

Листинг 3.21 – Реализация метода create

Метод смены режима изучения лексикона позволяет пользователю выбрать один из нескольких режимов обучения: перевод с испанского, перевод с русского или комбинированный вариант. Выбор режима осуществляется через выпадающий список (Dropdown). После изменения режима, обновленные настройки передаются на сервер с помощью метода **\*\*updateUserSettingsRequest\*\***, который сохраняет данные. Пользователь получает уведомление о успешном обновлении настроек.

### 3.8.20 Спросить ответ у искусственного интеллекта

Метод перевода текста с использованием ИИ реализован в методе `translateText` контроллера `DeepinfraController`. Реализация метода представлена на листинге 3.22.

```
@ApiBearerAuth()
@ApiOkResponse({ description: 'Translate text based on input data.'
})
@Post('translate')
public async translateText(
    @Body('task') task: string, // текст для перевода
    @Body('translate') translate: string, // переведенный текст
    @Body('targetLanguage') targetLanguage: string, // целевой язык
): Promise<any> {
    if (!task || !targetLanguage) {
        throw new Error('Input text and target language are required.');
```

Листинг 3.22 – Реализация метода `translateText`

Метод `translateText` позволяет пользователю перевести текст на целевой язык с использованием ИИ. Он принимает три параметра через тело запроса (`@Body()`). Метод проверяет, что обязательные параметры (`task` и `targetLanguage`) переданы. Если они отсутствуют, выбрасывается ошибка. После проверки метод вызывает функцию `translateText` из сервиса `DeepinfraService`, передавая текст для перевода, переведенный текст (если есть) и целевой язык. В результате метод возвращает переведенный текст.

### 3.8.21 Просмотр слова

Метод просмотра слов реализован в методе `getDictionariesReview` контроллера `DictionaryController`. Реализация метода представлена на листинге 3.23.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(DictionaryResponse))
@UseGuards(JwtAuthGuard)
@Get('review')
public getDictionariesReview(
    @Req() req: RequestWithUser,
```

```

): Promise<DictionaryReviewResponse[]> {
  return this.dictionaryService.getDictionaryReview(+req.user.id);
}

```

### Листинг 3.23 – Реализация метода getDictionaryReview

Метод `getDictionaryReview` предоставляет пользователю возможность просматривать доступные словари и информацию о прогрессе изучения слов. Данный метод защищен с помощью `JwtAuthGuard`, что гарантирует доступ только аутентифицированным пользователям. Если пользователь не прошел аутентификацию, запрос будет отклонен. После успешной аутентификации метод извлекает идентификатор пользователя (`user.id`) из запроса и передает его в сервис `DictionaryService`, вызывая функцию `getDictionaryReview`. В результате выполнения метода возвращается список словарей с детальной информацией о словах и текущем прогрессе пользователя в их изучении.

### 3.8.22 Перевод слов в соревновательном режиме

Метод перевода слов в соревновательном режиме реализован в методе `sendMessage` на стороне клиента, методе `create` контроллера `StatisticsController`, а также в методе `createOrUpdateStatistics` сервиса `StatisticsService`. Реализация методов представлена на листингах 3.24.

```

const sendMessage = async () => {
  setIsQuizButtonDisabled(true);

  try {
    if (
      socketWord.russianSpelling.toUpperCase() ===
      socketAnswer.toUpperCase()
    ) {
      const quizPoints = Number(timer);

      await statisticsApi.createOrUpdateStatistics({ quizPoints });
      socket.emit("correctAnswer", {
        name: user.getUser().name,
        quizPoints,
      });
      showSuccess(
        `Вы получили ${quizPoints} очков!`,
        "Правильный ответ!",
        2000
      );
    } else {
      showError(
        `Правильно: ${socketWord.englishSpelling} - ${socketWord.russianSpelling}`,
        "Ответ неверный :("
      );
    }
  } catch (error) {

```



```

        if (error?.response?.status === NOT_FOUND) {
            await loadData();
            showError(OLD_DATA);
        } else {
            showError(INVALID_INPUT);
        }
    }

    setSocketAnswer("");
};

```

Листинг 3.24 – Реализация метода sendMessage на фронтенде

Метод sendMessage обрабатывает отправку ответа пользователя при переводе слов в соревновательном режиме. Проверка ответа происходит с помощью метода сравнивающего введенный пользователем перевод (socketAnswer) с корректным переводом (socketWord.russianSpelling). При правильном ответе вычисляется количество очков (quizPoints) на основе оставшегося времени (timer). Отправляется запрос в statisticsApi.createOrUpdateStatistics для сохранения очков. Через socket.emit другие участники уведомляются о правильном ответе. Выводится сообщение об успешном выполнении задания. При неправильном ответе: выводится сообщение с правильным переводом слова.

Метод create контроллера StatisticsController, представленный на листинге 3.25, предназначен для обработки данных о прогрессе пользователя в соревновательном режиме.

```

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post()
private create(
    @Req() req: RequestWithUser,
    @Body() createStatisticDto: CreateStatisticDto,
) {
    return this.statisticsService.createOrUpdateStatistics(
        +req.user.id,
        createStatisticDto,
    );
}

```

Листинг 3.25 – Реализация метода create в контроллере

Метод create принимает объект CreateStatisticDto, содержащий количество набранных очков (quizPoints). Данные передаются в метод createOrUpdateStatistics сервиса StatisticsService, который обрабатывает их и сохраняет в базе данных

## 3.9 Реализация функций для роли «Администратор»

### 3.9.1 Изменение урока

Метод изменения существующего урока реализован в методе `updateLesson` контроллера `LessonController`. Реализация метода представлена в листинге 3.26

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(LessonResponse))
@UseGuards(JwtAuthGuard)
@Patch('/:id')
public updateLesson(
    @Param('id') id: string,
    @Body() updateLessonDto: UpdateLessonDto,
): Promise<LessonResponse> {
    return this.lessonService.updateLesson(+id, updateLessonDto);
}
```

Листинг 3.26 – Реализация метода `updateLesson`

Метод `updateLesson` позволяет пользователю изменить данные существующего урока. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод принимает `id` урока из параметров запроса и объект `UpdateLessonDto` из тела запроса. Затем вызывается функция `updateLesson` сервиса `LessonService`, которая обновляет данные урока. В ответе возвращается обновленный урок.

### 3.9.2 Добавление урока

Метод добавления нового урока реализован в методе `createLesson` контроллера `LessonController`. Реализация метода представлена в листинге 3.27.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post()
private createLesson(
    @Body() createLessonDto: CreateLessonDto,
): Promise<LessonResponse> {
    return this.lessonService.createLesson(createLessonDto);
}
```

Листинг 3.27 – Реализация метода `createLesson`

Метод `createLesson` позволяет пользователю добавить новый урок. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод принимает объект `CreateLessonDto` из тела запроса и передает его в сервис `LessonService`, вызывая функцию `createLesson`. В ответе возвращается созданный урок.

### 3.9.3 Удаление урока

Метод удаления урока реализован в методе `remove` контроллера `LessonController`. Реализация метода представлена в листинге 3.28.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('/:id')
public remove(@Param('id') id: string): Promise<void> {
    return this.lessonService.removeLesson(+id);
}
```

Листинг 3.28 – Реализация метода `remove`

Метод `remove` защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой. После успешной аутентификации метод принимает параметр `id` из URL, который представляет идентификатор удаляемого урока. Затем передает этот идентификатор в сервис `LessonService`, вызывая функцию `removeLesson`. В ответе не возвращается данных, однако урок успешно удаляется из базы данных.

### 3.9.4 Добавление задания

Метод добавления нового задания реализован в методе `createTask` контроллера `TaskController`. Реализация метода представлена в листинге 3.29.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(TaskResponse))
@UseGuards(JwtAuthGuard)
@Post()
private createTask(@Body() createTaskDto: CreateTaskDto):
    Promise<TaskResponse> {
    return this.taskService.createTask(createTaskDto);
}
```

Листинг 3.29 – Реализация метода `createTask`

Метод `createTask` позволяет пользователю добавить новое задание. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод принимает объект `CreateTaskDto` из тела запроса и передает его в сервис `TaskService`, вызывая функцию `createTask`. В ответе возвращается созданное задание.

### 3.9.5 Изменение задания

Метод изменения существующего задания реализован в методе `updateTask` контроллера `TaskController`. Реализация метода представлена в листинге 3.30.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(TaskResponse))
@UseGuards(JwtAuthGuard)
@Patch('/:id')
public updateTask(
    @Param('id') id: string,
    @Body() updateTaskDto: UpdateTaskDto,
): Promise<TaskResponse> {
    return this.taskService.updateTask(+id, updateTaskDto);
}
```

Листинг 3.30 – Реализация метода `updateTask`

Метод `updateTask` позволяет пользователю изменить данные существующего задания. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод принимает `id` задания из параметров запроса и объект `UpdateTaskDto` из тела запроса. Затем вызывается функция `updateTask` сервиса `TaskService`, которая обновляет данные задания. В ответе возвращается обновленное задание.

### 3.9.6 Удаление задания

Метод удаления задания реализован в методе `removeTask` контроллера `TaskController`. Реализация метода представлена в листинге 3.31.

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('/:id')
public removeTask(@Param('id') id: string): Promise<void> {
    return this.taskService.removeTask(+id);
}
```

Листинг 3.31 – Реализация метода `removeTask`

Метод `removeTask` позволяет пользователю удалить задание по его идентификатору. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод принимает `id` задания из параметров запроса и передает его в функцию `removeTask` сервиса `TaskService`. В результате выполнения задание удаляется.

### 3.9.7 Просмотр заданий

Метод просмотра всех заданий реализован в методе `getAllTasks` контроллера `TaskController`. Реализация метода представлена в листинге 3.32.

```
@ApiBearerAuth()
@ApiOkResponse(swaggerType(TaskResponse))
@UseGuards(JwtAuthGuard)
@Get()
public getAllTasks(): Promise<TaskResponse[]> {
  return this.taskService.getAllTasks();
}
```

Листинг 3.32 – Реализация метода `getAllTasks`

Метод `getAllTasks` позволяет пользователю получить список всех заданий. Данный метод защищен `JwtAuthGuard`, который проверяет, что запрос отправлен аутентифицированным пользователем. Если пользователь не аутентифицирован, запрос отклоняется с ошибкой.

После успешной аутентификации метод вызывает функцию `getAllTasks` сервиса `TaskService`, который обрабатывает логику получения всех заданий. В ответе возвращается массив объектов типа `TaskResponse`, каждый из которых содержит информацию о задании.

### 3.10 Структура клиентской части

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории `src`. Директории представлены в таблице 3.7

Таблица 3.7 – Основные директории проекта в папке `src` их назначение

Директория	Назначение
<code>api-requests</code>	Хранит файлы, связанные с запросами к API.
<code>components</code>	Содержит повторно используемые React-компоненты, такие как интерфейсные элементы ( <code>Input</code> , <code>NavBar</code> ), маршрутизаторы или другие строительные блоки приложения.
<code>pages</code>	Хранит страницы приложения
<code>stores</code>	Используется для управления состоянием приложения

styles	Хранит CSS стили проекта
utils	Хранит вспомогательные утилиты и функции, которые используются в разных частях приложения

Таблица соответствия маршрутов и страниц представлена в таблице 3.8.

Таблица 3.8 – Маршруты и страницы

Страница	Маршрут	Роль	Назначение страницы
Dictionary	/api/dictionary	Администратор, гость, пользователь	Просмотр, добавление, изменение, удаление своих словарей. Просмотр, добавление, изменение, удаление своих слов.

Продолжение таблицы 3.8

DictionaryLearn	/api/lexicon-progress	Пользователь	Выбор словаря для изучения, изучение лексикона, тренировка лексикона в соревновательном режиме
DictionaryReview	/api/dictionary/review	Пользователь	Просмотр словаря и слов Администратора, экспорт словаря Администратора, сортировка словаря по названию или описанию. Сортировка слов по испанскому переводу, транскрипции, русскому переводу, описанию.
Lesson	/api/lesson/admin	Администратор	Просмотр уроков, создание уроков, поиск уроков по названию, удаление уроков, изменение уроков, просмотр заданий урока, создание заданий, изменение заданий, удаление заданий, поиск задания по русскому и испанскому переводу.
LessonLearn	/api/grammar-progress	Пользователь	Просмотр уроков, выполнение заданий урока
Login	/api/auth/login	Гость	Авторизация Гостя
NotFound	/	Администратор, Пользователь, Гость	Страница для несуществующего маршрута
PersonalCabinet	/api/	Пользователь	Страница пользователя с его настройками.
Registration	/api/registration	Гость	Страница регистрации Гостя
Statistics	/api/statistics	Пользователь	Страница для просмотра статистики пройденных материалов.

Помимо маршрутов и страниц, приложение включает множество компонентов, которые обеспечивают функциональность и удобство использования клиентской части.

В таблице 3.9 представлено описание всех остальных компонентов приложения и их назначение.

Таблица 3.9 – Описание компонентов

Компонент	Назначение
AppRouter	Этот компонент отвечает за маршрутизацию в приложении. Распределяет доступ к страницам в зависимости от роли пользователя.
Navbar	Используется как навигационная панель по web-приложению.
Input	Универсальный входной элемент с валидацией, стилизованной ошибкой и гибкой настройкой.

### 3.11 Вывод по разделу

1. Определена программная платформа Node.js, СУБД PostgreSQL и объектно-реляционное отображение для реализации веб-приложения.

2. Определена структура серверной части, реализация функциональности для ролей, структура клиентской части.



## 4 Тестирование программного продукта

В этой главе рассмотрены и протестированы основные элементы интерфейса веб-приложения.

### 4.1 Функциональное тестирование для роли Пользователя

Таблица 4.1 – Описание тестирования функций Пользователя

Название функции	Описание тестирования	Итог тестирования функции
Просмотр своих словарей	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть» Ожидаемый результат: отображение списка словарей	Успешно.
Создание своих словарей	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», заполнить форму создания слова, нажать на кнопку «Добавить» Ожидаемый результат: создание и отображение созданного словаря	Успешно.
Изменение своего словаря	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на «карандаш», изменить данные в форме словаря Ожидаемый результат: изменение словаря.	Успешно.
Удаление своего словаря	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на «мусорный бак» у словаря Ожидаемый результат: удаление словаря.	Успешно.

Продолжение таблицы 4.1

Название функции	Описание тестирования	Итог тестирования функции
Просмотр слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на словарь Ожидаемый результат: отображение слов	Успешно.
Добавление слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на словарь, заполнить форму добавления слова Ожидаемый результат: добавление и отображение добавленного слова	Успешно.
Удаление слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на словарь, нажать на «мусорный бак» у слова Ожидаемый результат: удаление слова	Успешно.
Изменение слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на словарь, нажать на «карандаш», изменить данные в форме слова Ожидаемый результат: изменение слова	Успешно.
Экспорт слов	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на «Экспорт» рядом со словарем Ожидаемый результат: скачивание <i>JSON</i> файла со словами	Успешно.

Продолжение таблицы 4.1

Название функции	Описание тестирования	Итог тестирования функции
Импорт слов	<p>Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Изменить» или «Просмотреть», нажать на «Импорт» рядом со словарем, выбрать подходящий <i>JSON</i> файл</p> <p>Ожидаемый результат: добавление слов с <i>JSON</i> файла.</p>	Успешно.
Просмотр настроек аккаунта	<p>Действие: нажать на кнопку «Личный кабинет»</p> <p>Ожидаемый результат: переход на страницу настроек аккаунта</p>	Успешно.
Смена пароля	<p>Действие: нажать на кнопку «Личный кабинет», ввести в форму старый пароль, ввести новый пароль, ввести новый пароль повторно, нажать кнопку «Сменить пароль»</p> <p>Ожидаемый результат: переход на страницу настроек аккаунта</p>	Успешно.
Смена режима изучения лексики	<p>Действие: нажать на кнопку «Личный кабинет», в форме выбрать другой способ, нажать на кнопку «Сохранить»</p> <p>Ожидаемый результат: сохранение настроек в базе данных</p>	Успешно.
Просмотр уроков	<p>Действие: нажать на кнопку «Уроки»,</p> <p>Ожидаемый результат: отображение уроков на странице</p>	Успешно.
Просмотр теоретических сведений уроков	<p>Действие: нажать на кнопку «Уроки», нажать на нужный урок</p> <p>Ожидаемый результат: отображение теоретических сведений</p>	Успешно.

Продолжение таблицы 4.1

Название функции	Описание тестирования	Итог тестирования функции
Прохождение заданий	Действие: нажать на кнопку «Уроки», нажать на нужный урок, ответить на задания урока Ожидаемый результат: задание засчитано	Успешно.
Тренировка лексикона	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Учить», выбрать словарь, правильно перевести слово Ожидаемый результат: слово решено успешно.	Успешно.
Тренировка лексикона в соревновательном режиме	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Учить», правильно перевести слово на скорость, получить очки. В навигационной панели нажать на «Статистика» Ожидаемый результат: увидеть как решили другие.	Успешно.
Просмотр статистики	В навигационной панели нажать на «Статистика» Ожидаемый результат: увидеть статистику Пользователей.	Успешно.

## 4.2 Функциональное тестирование для роли Администратора

В таблице 4.2 представлено тестирование функций Администратора

Таблица 4.2 – Описание тестирования функций Администратора

Название функции	Описание тестирования	Итог тестирования функции
Просмотр своих словарей	Действие: нажать на кнопку «Словари и слова» в навигационной панели Ожидаемый результат: отображение списка словарей	Успешно.

Продолжение таблицы 4.2

Название функции	Описание тестирования	Итог тестирования функции
Создание своих словарей	Действие: нажать на кнопку «Словари и слова» в навигационной панели, заполнить форму создания слова, нажать на кнопку «Добавить» Ожидаемый результат: создание и отображение созданного словаря	Успешно.
Изменение своего словаря	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «карандаш», изменить данные в форме словаря Ожидаемый результат: изменение словаря.	Успешно.
Удаление своего словаря	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «мусорный бак» у словаря Ожидаемый результат: удаление словаря.	Успешно.
Просмотр слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на словарь Ожидаемый результат: отображение слов	Успешно.
Добавление слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на словарь, заполнить форму добавления слова Ожидаемый результат: добавление и отображение добавленного слова	Успешно.
Удаление слова	Действие: нажать на кнопку «Словари и слова», нажать на словарь, нажать на «мусорный бак» у слова Ожидаемый результат: удаление слова	Успешно.
Изменение слова	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на словарь, нажать на «карандаш», изменить данные в форме слова Ожидаемый результат: изменение слова	Успешно.

Продолжение таблицы 4.2

Название функции	Описание тестирования	Итог тестирования функции
Экспорт слов	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Экспорт» рядом со словом Ожидаемый результат: скачивание <i>JSON</i> файла со словами	Успешно.
Импорт слов	Действие: нажать на кнопку «Словари и слова» в навигационной панели, нажать на «Импорт» рядом со словом, выбрать подходящий <i>JSON</i> файл Ожидаемый результат: добавление слов с <i>JSON</i> файла.	Успешно.
Добавление урока	Действие: нажать на кнопку «Уроки» в навигационной панели, заполнить форму для создания урока Ожидаемый результат: создание урока.	Успешно.
Удаление урока	Действие: нажать на кнопку «Уроки» в навигационной панели, нажать на кнопку «Мусорки» рядом с уроком Ожидаемый результат: удаление урока.	Успешно.
Просмотр задания	Действие: нажать на кнопку «Уроки» в навигационной панели, нажать на урок Ожидаемый результат: получение заданий урока.	Успешно.
Удаление задания	Действие: нажать на кнопку «Уроки» в навигационной панели, нажать на урок, нажать на кнопку «Мусорки» рядом с уроком. Ожидаемый результат: удаление задания.	Успешно.
Добавление задания	Действие: нажать на кнопку «Уроки» в навигационной панели, нажать на урок, заполнить форму для добавления заданий, нажать кнопку «Добавить» Ожидаемый результат: создание задания.	Успешно.

### 4.3 Функциональное тестирование для роли Гостя

В таблице 4.3 представлены результаты тестирования функций Гостя

Таблица 4.3 - Описание тестирования функций Гостя

Название функции	Описание тестирования	Итог тестирования функции
Регистрация	Действие: ввести корректные данные(имя, фамилию, <i>email</i> , пароль) и отправить форму. Проверь письмо с паролем на почте Ожидаемый результат: учетная запись создана, пользователь получил письмо на почту.	Успешно.
Авторизация	Действие: ввести корректные данные( <i>email</i> , пароль) и отправить форму. Ожидаемый результат: пользователь входит в учетную запись .	Успешно.

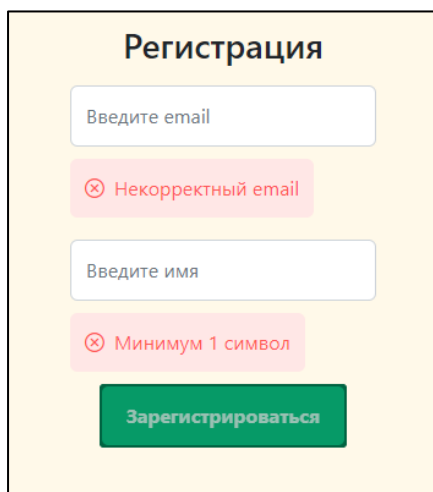
### 4.4 Вывод по разделу

1. Выполнено функциональное тестирование для каждой функции, представленной в диаграмме вариантов использования.
2. Все этапы пройдены успешно, web-приложение работает стабильно.
3. Количество тестов составило 39, покрытие тестами – 95%.

## 5 Руководство пользователя

### 5.1 Регистрация

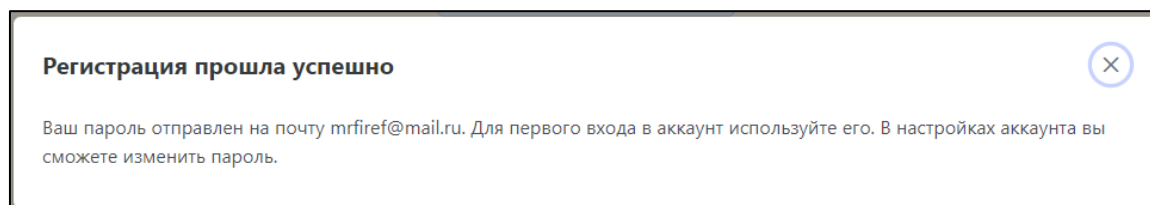
При первом переходе к приложению у неавторизованного пользователя есть возможность зарегистрироваться либо авторизоваться. Форма со страницы регистрации приведена на рисунке 5.1.



The registration form is titled "Регистрация" (Registration). It contains two input fields: "Введите email" (Enter email) and "Введите имя" (Enter name). Below the email field is a red error message: "✗ Некорректный email" (Incorrect email). Below the name field is a red error message: "✗ Минимум 1 символ" (Minimum 1 symbol). At the bottom of the form is a green button labeled "Зарегистрироваться" (Register).

Рисунок 5.1 – Форма для регистрации пользователя

При заполнении формы корректными данными и последующем нажатии на кнопку регистрации, пользователь получит сообщение, показанное на рисунке 5.2.



The success message dialog box is titled "Регистрация прошла успешно" (Registration was successful). It contains the text: "Ваш пароль отправлен на почту mrfiref@mail.ru. Для первого входа в аккаунт используйте его. В настройках аккаунта вы сможете изменить пароль." (Your password has been sent to the email mrfiref@mail.ru. For the first login, use it. In the account settings, you will be able to change the password.) There is a close button (✗) in the top right corner.

Рисунок 5.2 – Сообщение об успешной регистрации

После закрытия диалогового окна пользователь перенаправляется на страницу входа, представленную на рисунке 5.3.

### 5.2 Авторизация

После успешной регистрации требуется выполнить вход на сайт, необходимо корректно заполнить данные email и пароль. Форма входа в аккаунт представлена на рисунке 5.3



**Авторизация**

mrfiref@mail.ru

.....

**Войти**

Рисунок 5.3 – Форма для авторизации

После успешного входа пользователь будет отправлен на страницу просмотра статистики.

### 5.3 Просмотр своих словарей

Функционал доступный Пользователю и Администратору, для этого необходимо выбрать соответствующий раздел в меню. У Администратора это меню будет раздел «Словари». У Пользователя это будет подраздел «Изменить» в разделе «Словари и слова». На рисунке 5.4 изображена страница со словарями.

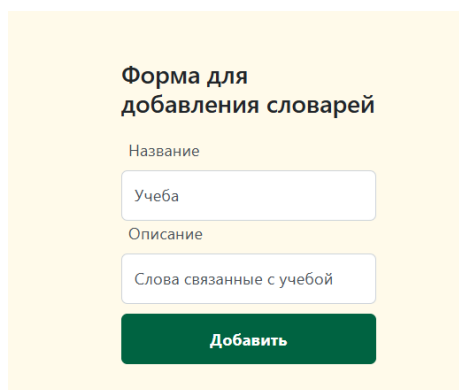
Словари				Слова			
название	описание	импорт	экспорт	испанский	транскрипция	русский	описание
Технологии				aplicación	a.pli.ka. ðjon	приложение	Программное обеспечение для выполнения конкретных задач
Дом	Слова на тему дома			pantalla	pan.'ta.ʃa	экран	Устройство для отображения информации
Животные	Слова для животных			servidor	ser.'βiðor	сервер	Компьютер, предоставляющий услуги другим компьютерам в сети
				ordenador	or.ðe.na'ðor	компьютер	Испанский аналог слова 'компьютер'
				red	red	сеть	Группа взаимосвязанных устройств, обеспечивающих обмен информацией

Рисунок 5.4 – Страница просмотра своих словарей

После выбора раздела будет представлена форма со словарями и словами в них. А также две формы для добавления и изменения для словарей и слов.

### 5.4 Добавление словаря

На странице описанной выше так-же можно добавить словарь, для этого необходимо пролистать страницу чуть ниже и увидеть форму для заполнения. Форма представлена на рисунке 5.5



**Форма для добавления словарей**

Название

Учеба

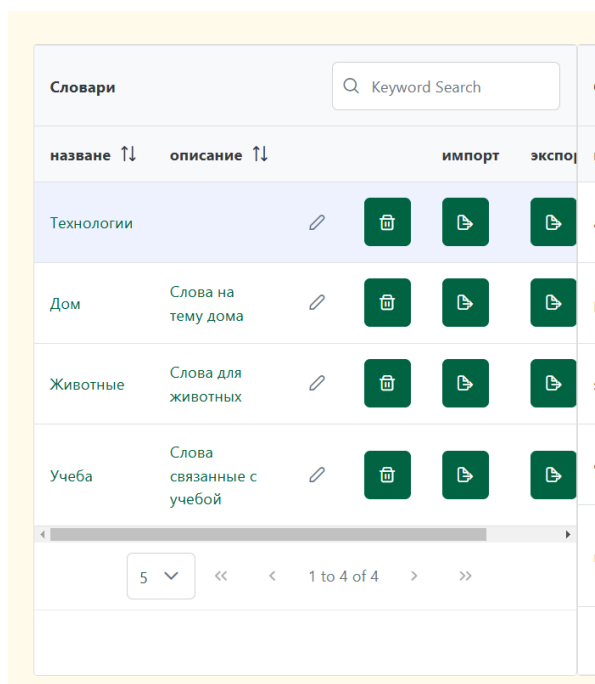
Описание

Слова связанные с учебой

**Добавить**

Рисунок 5.5 – Форма для добавления словарей

В данной форме необходимо указать название и описание словаря, после этого необходимо нажать на кнопку «Добавить». На рисунке 5.6 будет изображено как изменилась форма с имеющимися словарями



Словари		Keyword Search		
название ↑↓	описание ↑↓		импорт	экспорт
Технологии				
Дом	Слова на тему дома			
Животные	Слова для животных			
Учеба	Слова связанные с учебой			

5 1 to 4 of 4

Рисунок 5.6 – Словарь добавился в форму

### 5.5 Удаление своего словаря

На этой-же странице моно удалить созданный словарь, для этого необходимо на форме нажать на кнопку с изображением корзины рядом со словарем. На рисунке 5.7 отображена форма в момент когда пользователь навелся на кнопку удаления.

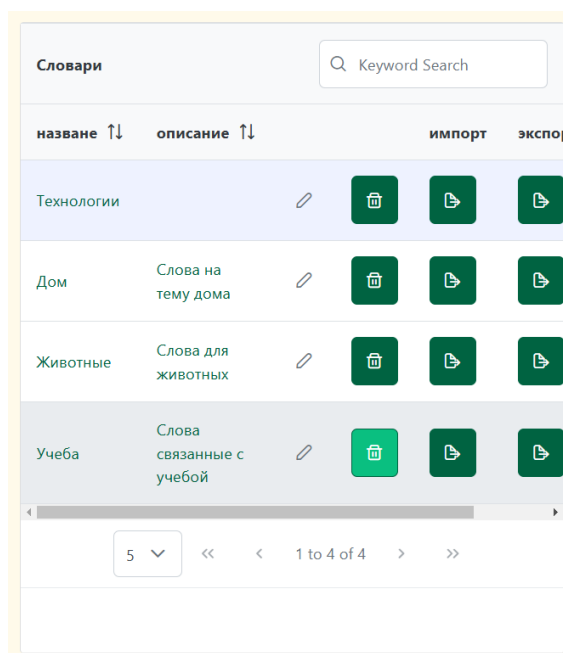


Рисунок 5.7 – Удаление словаря

После удаления словаря, он пропадет с формы.

## 5.6 Изменение своего словаря

Для изменение словаря необходимо выбрать кнопку с изображением карандаша рядом со словарем. После ее нажатия в форме появится возможность ввести новое название и описание для словаря. На рисунке 5.8 изображена форма со словарями после нажатия на кнопку изменения.

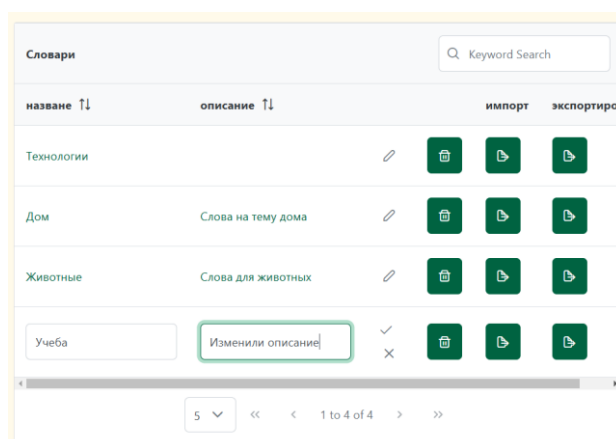


Рисунок 5.8 –Изменение описания словаря

После нажатия на кнопку для изменения, вместо нее появляется 2 кнопки. Первая для принятия изменений, вторая для отмены изменений. На рисунке 5.9 изображено как изменилось описание после его редактирования и принятия изменений.

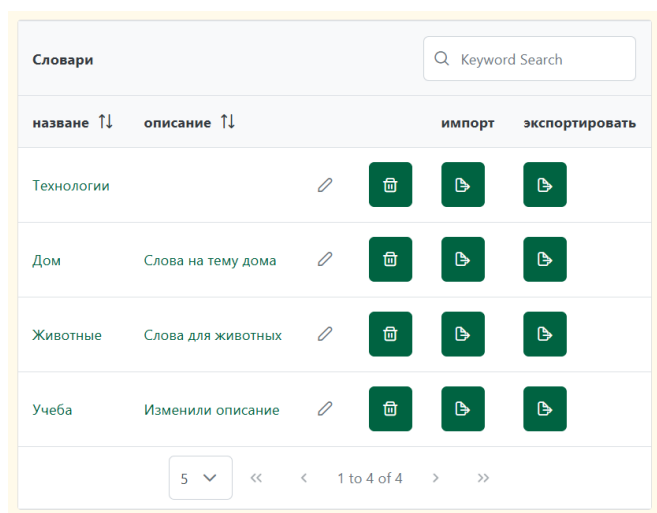


Рисунок 5.9 –Измененное описание словаря

Как можно видеть словарь изменился успешно и изменений словарь отображается корректно.

### 5.7 Добавление своего слова

Для добавления слова нам необходимо выбрать нужный словарь, как отображено на рисунке 5.10.

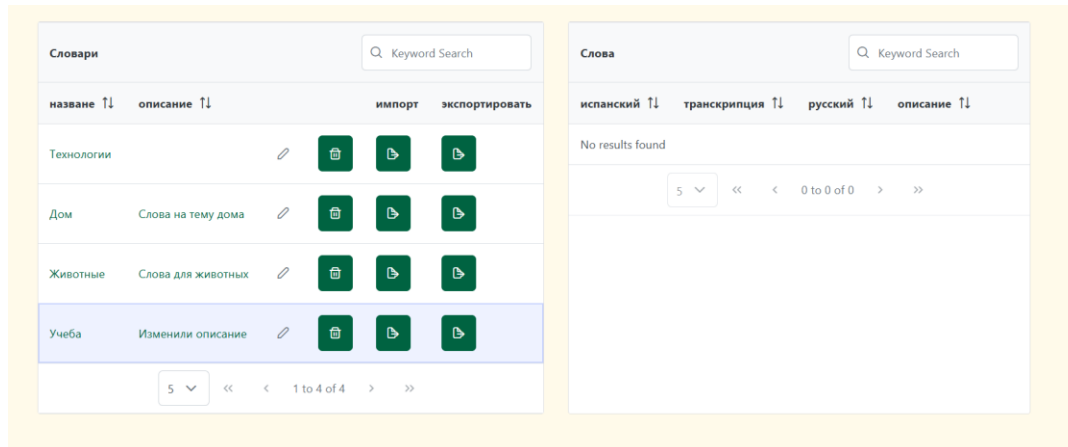
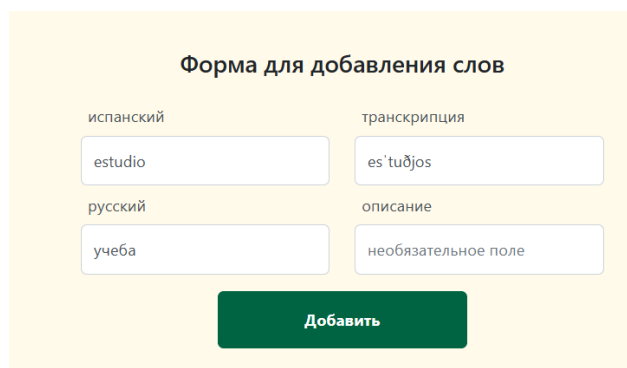


Рисунок 5.10 – Выбор словаря

После выбора словаря мы можем в форме ниже добавить слово, для этого необходимо заполнить поле с написанием слова на испанском языке, его транскрипция, а так-же его перевод на русский. Также можно заполнить поле с описанием. На рисунке 5.11 изображена заполненная форма слова.

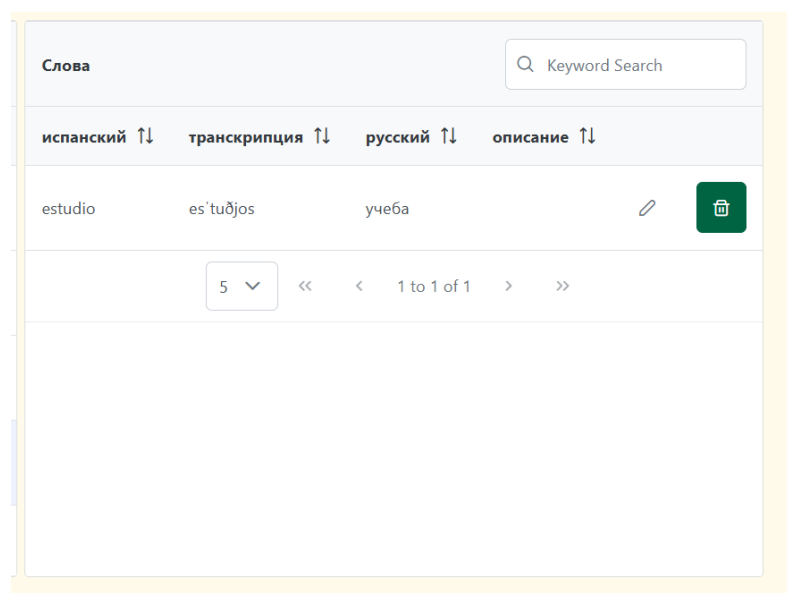


**Форма для добавления слов**



испанский	транскрипция
<input type="text" value="estudio"/>	<input type="text" value="es'tuðjos"/>
русский	описание
<input type="text" value="учеба"/>	<input type="text" value="необязательное поле"/>

Рисунок 5.11 – Форма для добавления слова

После заполнения формы необходимо нажать на кнопку «Добавить», на рисунке 5.12 изображено добавленное слово в словаре.



**Слова**

испанский ↑↓	транскрипция ↑↓	русский ↑↓	описание ↑↓
estudio	es'tuðjos	учеба	 

5 ▾ << < 1 to 1 of 1 > >>

Рисунок 5.12 – Добавленное слово

Как можно видеть слово успешно создано и добавилось в словарь.

## 5.8 Изменение своего слова

Для изменение слова необходимо выбрать кнопку с изображением карандаша рядом со словом. На рисунке 5.13 изображена форма со словами после нажатия на кнопку изменения.

Слова

Keyword Search

транскрипция ↑↓ русский ↑↓ описание ↑↓

es'tuðjos учеба

✓ ✕

5 << < 1 to 1 of 1 > >>

Рисунок 5.13 – Изменение слова

После ее нажатия в форме появится возможность ввести все параметры слова с формы. Так-же вместо кнопки изменения появляется две кнопки, для принятия изменений и для отмены изменений. На рисунке 5.14 изображено как изменилось слово после изменения описания.

Слова

Keyword Search

испанский ↑↓ транскрипция ↑↓ русский ↑↓ описание ↑↓

estudio es'tuðjos учеба Тестовое описание

✎

5 << < 1 to 1 of 1 > >>

Рисунок 5.14 –Измененное слово

Как видно слово успешно изменилось и корректно отображается в форме.

## 5.9 Удаление своего слова

В этой же форме можно удалить созданное слово, для этого необходимо на форме нажать на кнопку с изображением корзины рядом со словом. На рисунке 5.15 отображена форма в момент когда пользователь навелся на кнопку удаления.

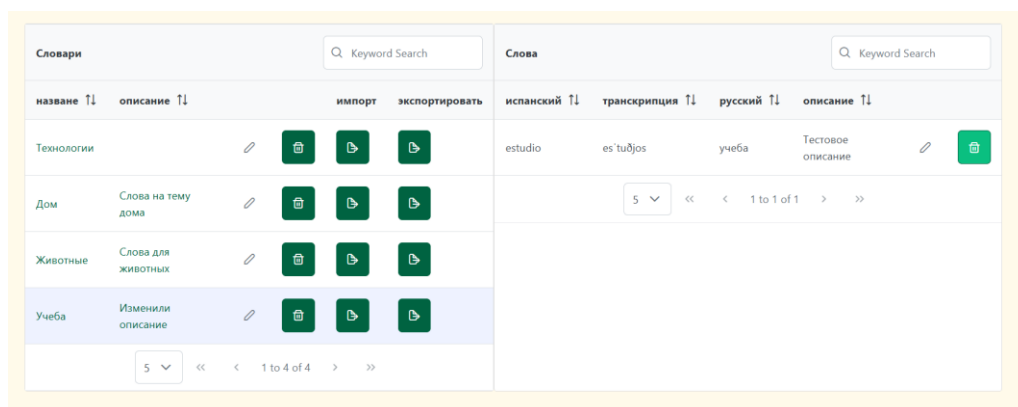


Рисунок 5.15 –Удаление слова

На рисунке 5.16 можно увидеть что слово пропало с формы.

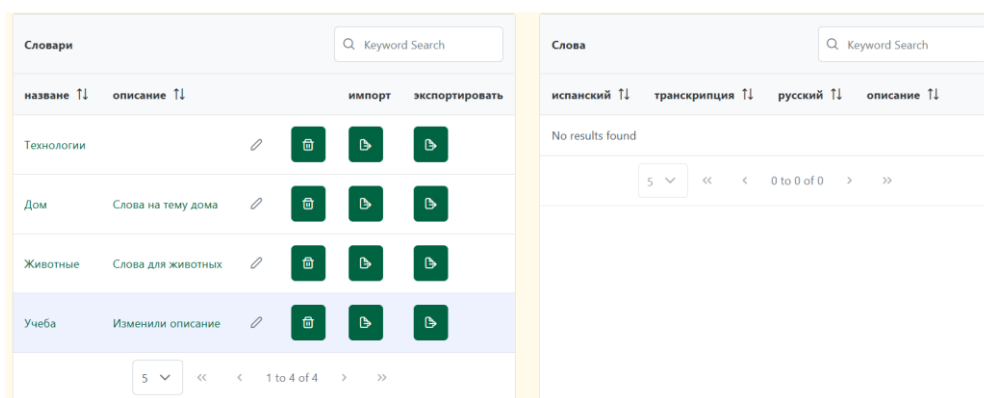


Рисунок 5.16 – Форма после удаления слово

Как видно функция удаления слова работает корректно.

## 5.10 Импорт слов

Также на этой форме можно произвести импорт слов в словарь, на рисунке 5.17 отображена кнопка для импорта в момент наведения на нее пользователя.

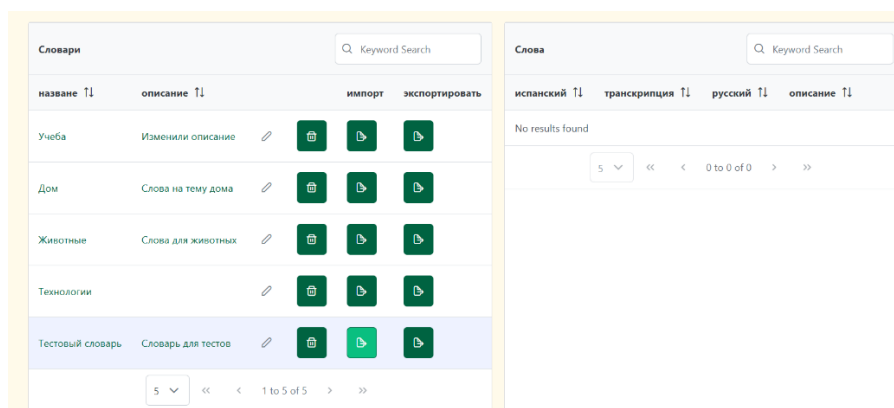


Рисунок 5.17 – Кнопка импорт слов

После нажатия на кнопку откроется окно для выбора файла. Необходимо выбрать файл в формате json. На рисунке 5.18 изображен пример выбора файла.

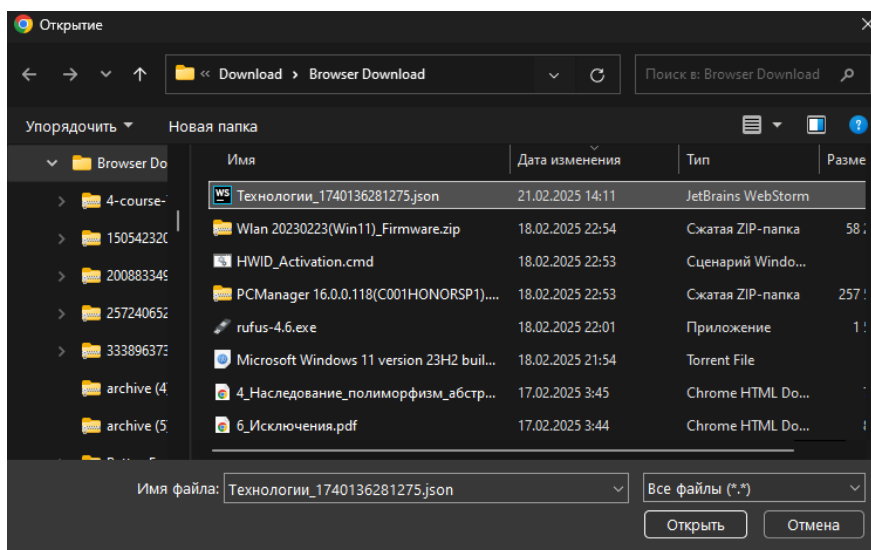


Рисунок 5.18 – Файл со словами

На рисунке 5.19 изображен процесс импорта слов в словарь.

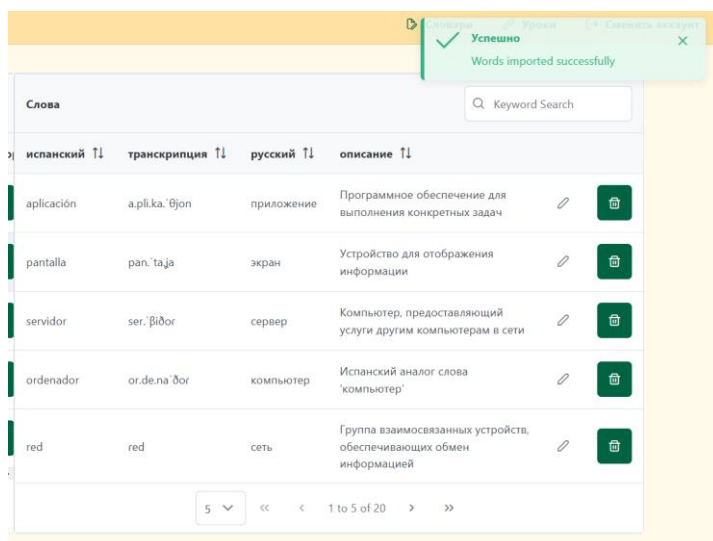


Рисунок 5.19 – Успешный импорт слов

Если слова со словаря совпадают со словами из импортируемого файла то добавляются только новые для словаря слова.

## 5.11 Экспорт слов

В форме есть кнопка для экспорта слов в формате json, на рисунке 5.20 отображено как выглядит эта кнопка в момент наведения.



Словари				Слова			
название ↑↓	описание ↑↓	импорт	экспортировать	испанский ↑↓	транскрипция ↑↓	русский ↑↓	описание ↑↓
Технологии				animal	a.ni.'mal	животное	Существа помимо человека
Дом	Слова на тему дома			perro	'pe.rro	собака	Дом. живот., друг человека
Животные	Слова для животных			gato	'ga.to	кот	Животное в сапогах
Учеба	Изменили описание			pajaro	'pa.xa.ro	птица	Пернатое летающее животное
				pez	'pes	рыба	Водное позвоночное животное

Рисунок 5.20 –Кнопка экспорта слов

После нажатия на данную кнопку происходит скачивание файла в формате json, на рисунке 5.21 изображен экспортируемый файл.

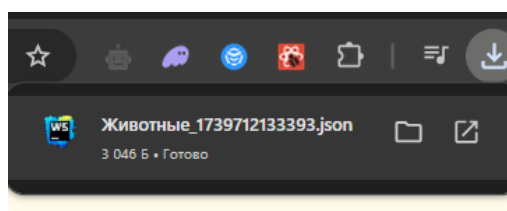


Рисунок 5.21 – Файл с экспортируемыми словами

На рисунке 5.22 изображено содержимое экспортируемого файла, такой-же формат используется и для импорта.

```

1  [
2  {
3    {
4      "spanishSpelling": "animal",
5      "transcription": "a.ni.'mal",
6      "russianSpelling": "животное",
7      "description": "Существа помимо человека"
8    },
9    {
10     "spanishSpelling": "perro",
11     "transcription": "'pe.rro",
12     "russianSpelling": "собака",
13     "description": "Дом. живот., друг человека"
14   },
15   {
16     "spanishSpelling": "gato",
17     "transcription": "'ga.to",
18     "russianSpelling": "кот",
19     "description": "Животное в сапогах"
20   },
21   {
22     "spanishSpelling": "pajaro",
23     "transcription": "'pa.xa.ro",
24     "russianSpelling": "птица",
25     "description": "Пернатое летающее животное"
26   },
27   {
28

```

Рисунок 5.22 – Содержимое экспортируемого файла

Формат экспортируемого файла идентичен используемому при импорте слов. При импорте система автоматически проверяет корректность данных и добавляет новые слова в словарь пользователя либо уведомляет о найденных ошибках.

## 5.12 Просмотр словарей Администратора

Функция Пользователя для просмотра словарей и слов в них созданных Администратором. Для перехода необходимо выбрать подраздел «Просмотреть» в разделе «Словари и слова». На рисунке 5.23 изображен выбор нужного подраздела.

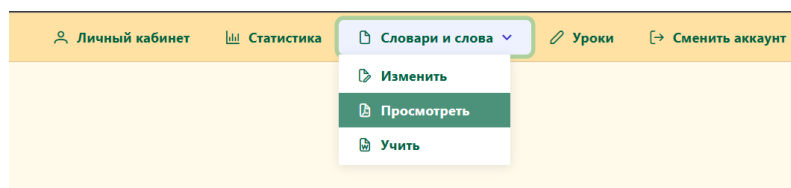


Рисунок 5.23 – Кнопка перехода к разделу просмотра словарей Администратора

После нажатия на данный подраздел можно просмотреть словари

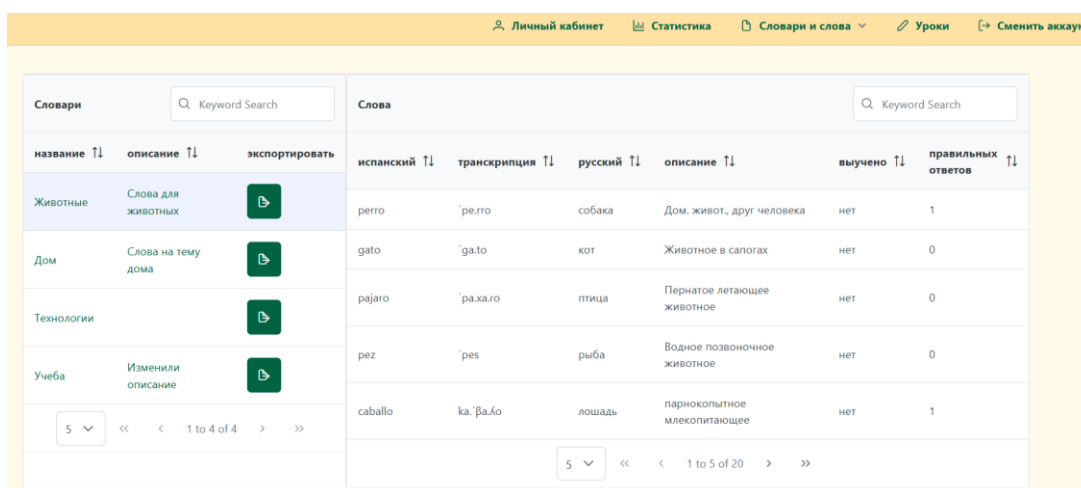


Рисунок 5.24 – Просмотр словарей

На странице отображены все словари Администратора, можно просмотреть слова словарей и экспортировать их.

## 5.13 Просмотр статистики

Для просмотра статистики необходимо перейти в раздел «Статистика». На рисунке 5.25 отображена страница статистики.

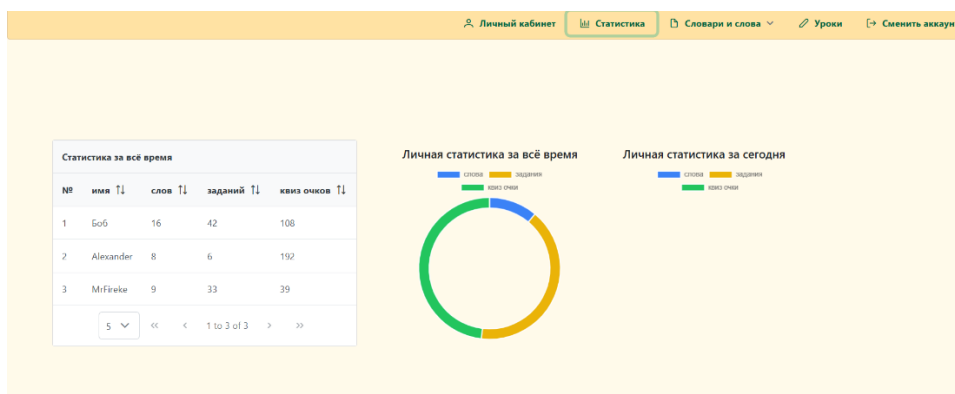


Рисунок 5.25 – Просмотр статистики

На странице можно увидеть количество выученных слов, количество пройденных заданий а так-же очки которые Пользователь получает за обучение лексике в соревновательном режиме.

### 5.14 Решение заданий

Функция Пользователя, для прохождения заданий необходимо выбрать раздел Уроки. В этом разделе необходимо выбрать урок для прохождения. На рисунке 5.26 будет отображена страница уроков с формой для выбора урока и для прохождения задания

Личный кабинет | Статистика | Словари и слова | **Уроки** | Сменить аккаунт

Уроки

Keyword Search

название

Слова

/Дом

1 to 1 of 1

Переведите предложение, расставляя слова в нужном порядке или используйте голосовой ввод

Вы уже выполняли это задание верно

Сейчас выполнено заданий: 0/6. Из них верно: 0

Мой дом большой и уютный

Mi grande acogedora casa y es

Применить результат

Результат: Спросить

Рисунок 5.26 – Форма задания и уроков

Для решения задания необходимо в правильном порядке указать слова в предложении. Так-же можно ввести данные слова с помощью голосового ввода. Также можно спросить ответ и объяснение у ИИ. На рисунке 5.27 будет отображен результат прохождения урока.

**Переведите предложение, расставляя слова в нужном порядке или используйте голосовой ввод**

Вы уже выполняли это задание верно

Сейчас выполнено заданий: 1/6. Из них верно: 1

На кухне есть все необходимое для приготовления пищи

Результат:

Рисунок 5.27 – Решенное задание

Тут отображено сколько заданий пользователь решил и сколько из них верные. При неправильном ответе появится уведомление что задание решено неверно, и показывает правильный ответ.

### 5.15 Просмотр уроков

Функция Пользователя и Администратора, для просмотра доступных уроков необходимо перейти в раздел «Уроки» у Пользователя и у Администратора. На рисунке 5.28 изображена форма с доступными уроками, которые сортируются по темам.

**Уроки**

название ↑↓

▼

Слова

Дом

1 to 1 of 1

Рисунок 5.28 – Форма уроков

На рисунке изображена форма с уроками. «Слова» это тема урока, и сам урок «Дом»

## 5.16 Просмотр настроек аккаунта

Функция Пользователя, для просмотра настроек аккаунта необходимо перейти в раздел «Личный кабинет». На рисунке 5.29 изображено содержимое страницы с настройками аккаунта.

Рисунок 5.29 – Настройки аккаунта

На странице есть возможность сменить пароль, поменять настройки аккаунта.

## 5.17 Смена пароля

Форма смены пароля доступна на странице настроек аккаунта. Форма для смены изображено на рисунке 5.30

Рисунок 5.30 – Смена пароля

Для смены пароля необходимо ввести текущий пароль, новый пароль и снова ввести новый пароль. На рисунке 5.31 показан результат смены пароля.

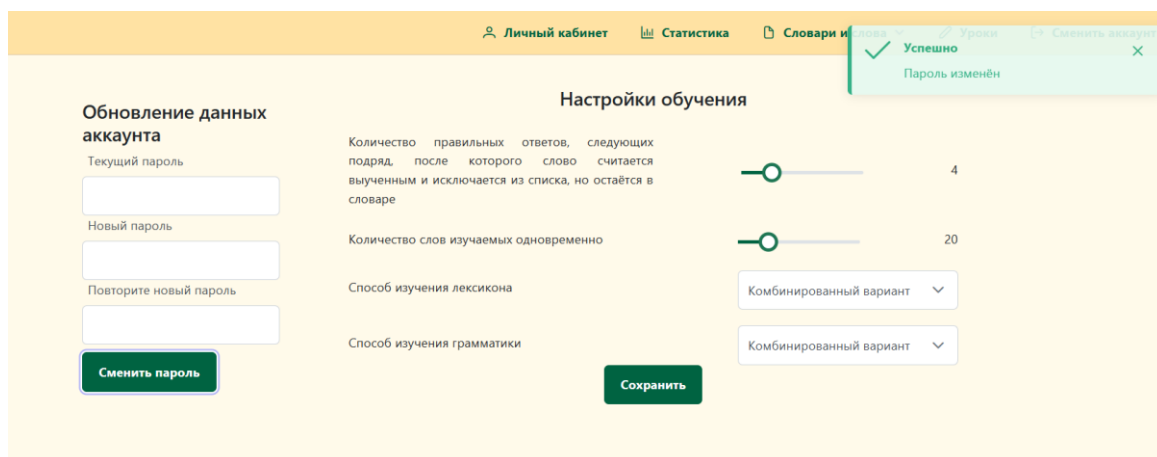


Рисунок 5.31 – Уведомление о успешной смене пароля

После смены пароля появляется уведомление которое указывает на то что пароль изменен, если пароль не изменился то будет соответствующее уведомление.

### 5.18 Смена количества правильных ответов

Данная функция Пользователя находится на странице «Личный кабинет», для смены количества правильных ответов после которого слово считается выученным, необходимо передвинуть соответствующий ползунок. На рисунке 5.32 изображен результат изменения настроек.

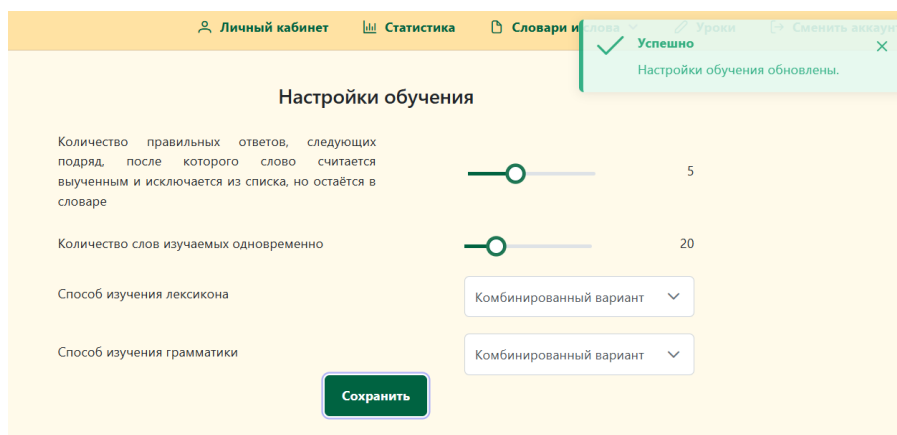


Рисунок 5.32 – Смена количества правильных ответов

После изменения настроек появляется уведомление об успешной смене настроек.

### 5.19 Смена режима изучения грамматики

Данная функция Пользователя находится на странице «Личный кабинет», для смены режима изучения грамматики необходимо выбрать с какого языка будет производиться перевод. На рисунке 5.33 изображен выбор способа изучения грамматики.

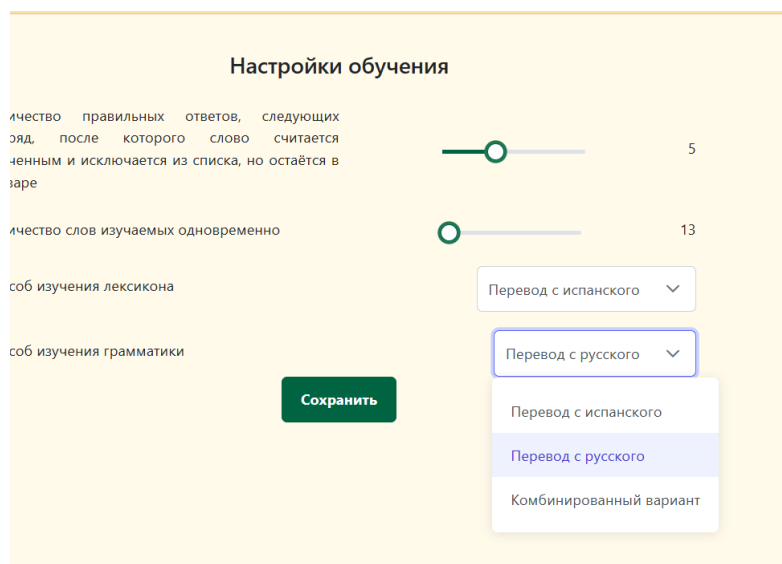


Рисунок 5.33 – Выбор способа изучения грамматики

После выбора необходимо нажать кнопку «Сохранить». На рисунке 5.34 изображено уведомление о успешной смене

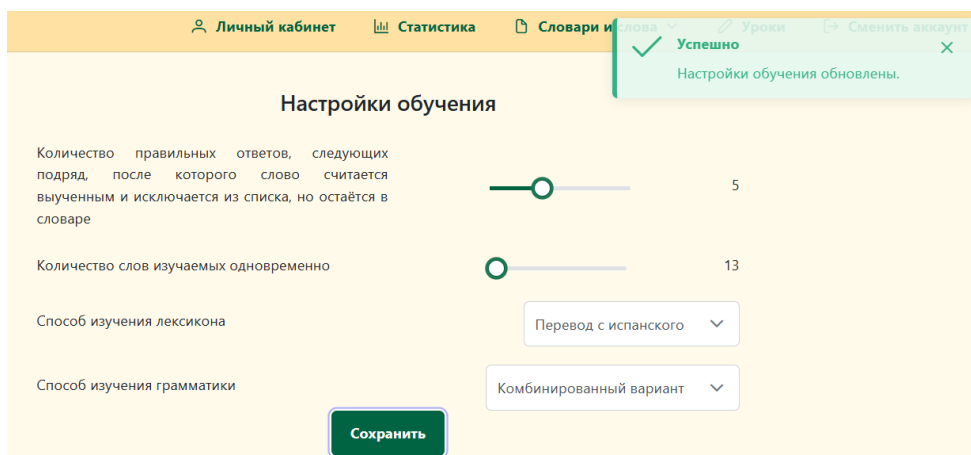


Рисунок 5.34 – Уведомление о успешной смене режима изучения грамматики

После изменения настроек появляется уведомление об успешной смене настроек.

## 5.20 Смена количества изучаемых слов

Данная функция Пользователя находится на странице «Личный кабинет» и позволяет изменить количество изучаемых слов. На рисунке 5.35 изображено изменение количества изучаемых слов и уведомление об успешном обновлении настроек обучения.

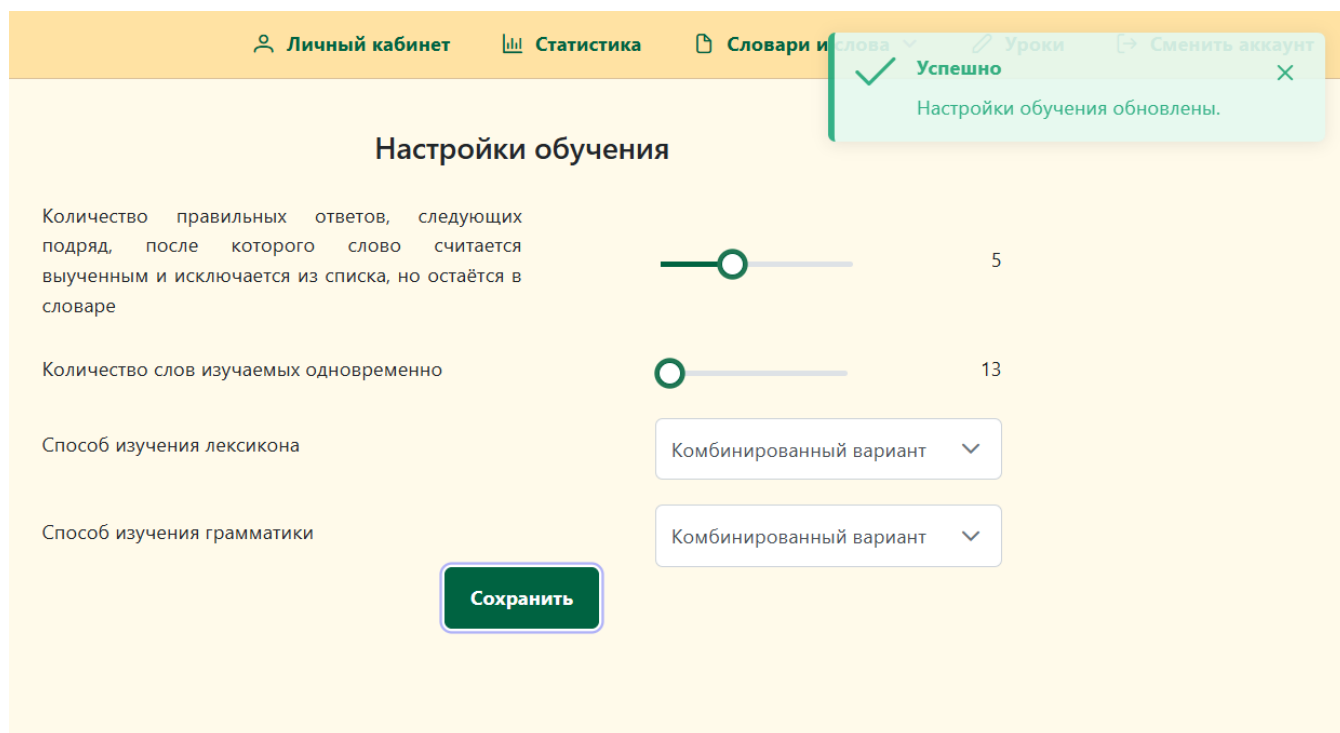


Рисунок 5.35 – Смена количества изучаемых слов

На данной странице пользователь может выбрать нужное количество слов для изучения.

### 5.21 Смена режима изучения лексикона

Данная функция Пользователя находится на странице «Личный кабинет» и позволяет изменить режим изучения лексикона. На рисунке 5.36 изображены режимы изучения лексикона.

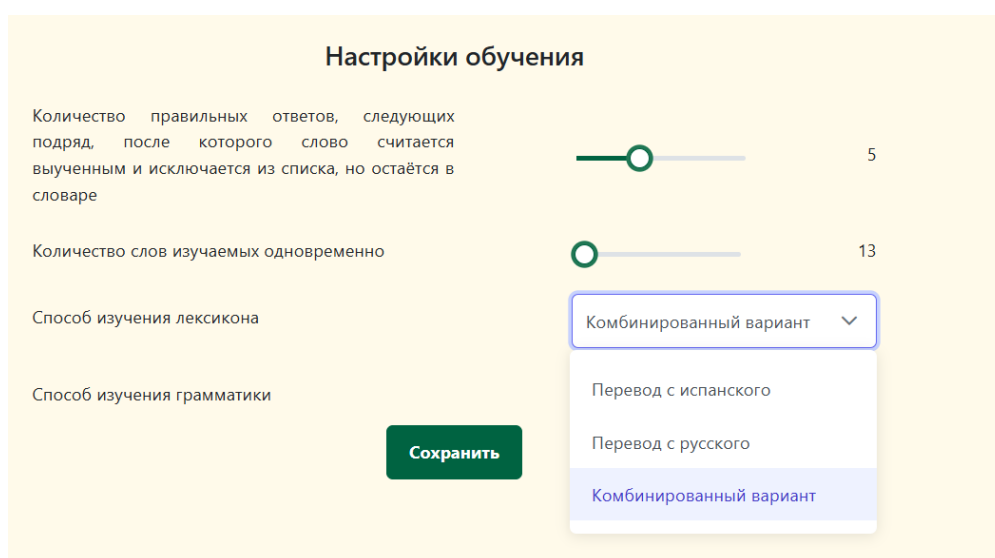


Рисунок 5.36 – Смена режима изучения слов



После выбора необходимо нажать кнопку «Сохранить». На рисунке 5.37 изображено уведомление о успешном изменении настроек обучения.

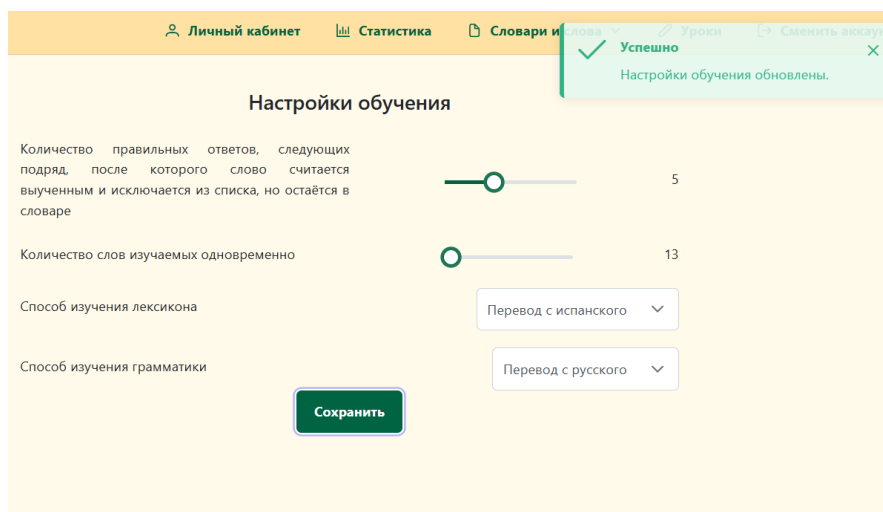


Рисунок 5.37 – Уведомление о успешной смене количества изучаемых слов

На данной странице пользователь может выбрать для себя удобный способ изучения.

## 5.22 Просмотр слова

Данная функция Пользователя и Администратора служит для просмотра слов словаря, для нахождения данной функции на странице необходимо перейти в подраздел «Изменить» в раздел «Словари и слова» у Пользователя. Для Администратора необходимо перейти в раздел «Словари». На рисунке 5.38 изображена форма со словарями и словами в них.

Словари			Слова						
название ↑↓	описание ↑↓	экспортировать	испанский ↑↓	транскрипция ↑↓	русский ↑↓	описание ↑↓	выучено ↑↓	правильных ответов ↑↓	
Животные	Слова для животных		perro	'pe.rro	собака	Дом. живот., друг человека	нет	1	
Дом	Слова на тему дома		gato	'ga.to	кот	Животное в сапогах	нет	0	
Технологии			pajaro	'pa.xa.ro	птица	Пернатое летающее животное	нет	0	
Учеба	Изменили описание		pez	'pes	рыба	Водное позвоночное животное	нет	0	
			caballo	ka.'ba.ʎo	лошадь	парнокопытное млекопитающее	нет	1	

Рисунок 5.38 – Просмотр слов

Для просмотра слов словаря необходимо нажать на нужный словарь.

## 5.24 Спросить ответ у искусственного интеллекта

Функция Пользователя, при решении заданий появляется кнопка «Спросить ИИ». На рисунке 5.39 изображено всплывающее окно с ответом от искусственного интеллекта.

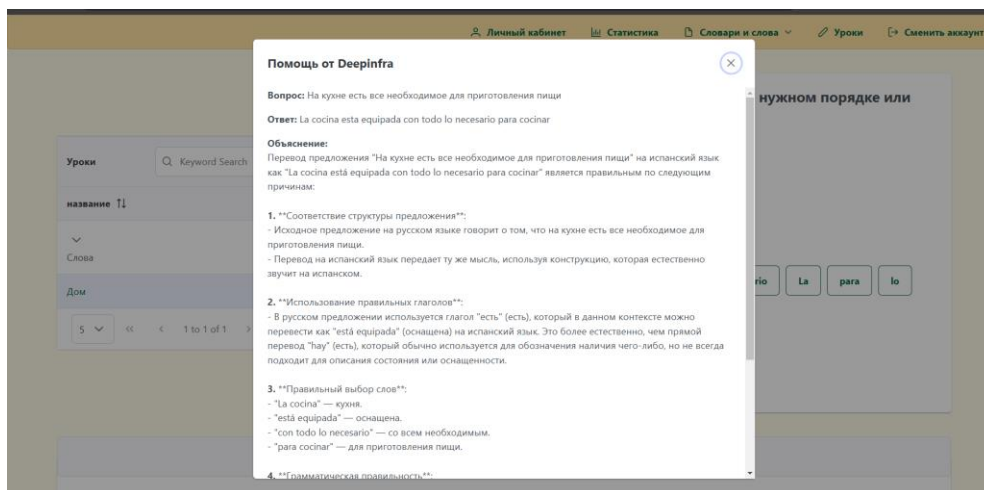


Рисунок 5.39 – Спросить ответ у искусственного интеллекта

В всплывающем окне будет расписан сам вопрос, правильный ответ и пояснение к заданию почему именно так переводится предложение.

## 5.25 Перевод слов в соревновательном режиме

Данная функция Пользователя расположена в подразделе «Учить» раздела «Словари и слова». На рисунке 5.40 изображена форма для перевода слов в соревновательном режиме.

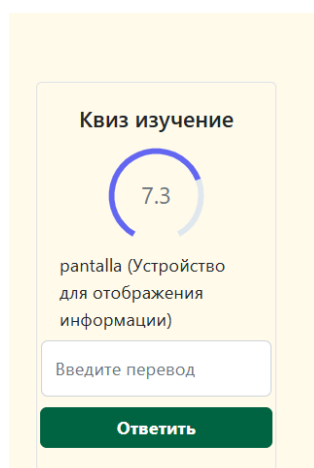


Рисунок 5.40 – Перевод слов в соревновательном режиме

Для перевода слова необходимо за определенный промежуток времени перевести слово и нажать кнопку «Ответить». На рисунке 5.41 отображено уведомление о правильном ответе.

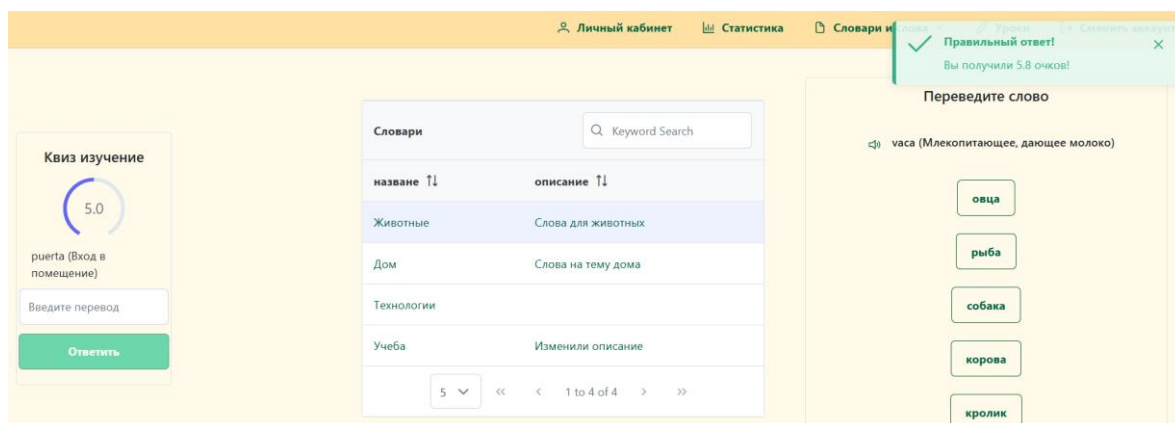


Рисунок 5.41 – Уведомление о правильном ответе

В уведомлении прописывается сколько очков заработал пользователь. Чем быстрее пользователь написал верный ответ, тем больше очков.

## 5.27 Изменение урока

Функция Администратора которая расположена в разделе «Уроки». При нажатии на урок его тема, название и теория появляется в форме ниже списка уроков. На рисунке 5.42 отображена форма для добавления и изменения уроков.

 The screenshot shows a form titled 'Тестовый урок' with a trash icon. Below the title is a pagination bar showing '5' and '1 to 2 of 2'. The main section is titled 'Форма для добавления тем и/или уроков'. It contains three input fields: 'Название темы' (containing 'Тестовый урок'), 'Название урока' (containing 'Тестовый урок'), and 'Теория к уроку' (containing 'Тестовый урок'). At the bottom, there are two green buttons: 'Изменить' and 'Добавить'.

Рисунок 5.42 – Форма для изменения и добавления уроков

Можно изменить все параметры урока. На рисунке 5.43 изображены измененные параметры урока после изменения.

Рисунок 5.43 – Уведомление о правильном ответе

Как видно из изображения название успешно поменялось, так-же поменялась тема и теория.

## 5.28 Добавление урока

Функция Администратора которая расположена в разделе «Уроки». На рисунке 5.44 изображена форма для добавления тем и уроков с заполненными полями.

Рисунок 5.44 – Форма для добавления тем и уроков

После заполнения формы необходимо нажать на кнопку «Добавить». На рисунке 5.45 изображен добавленный урок.

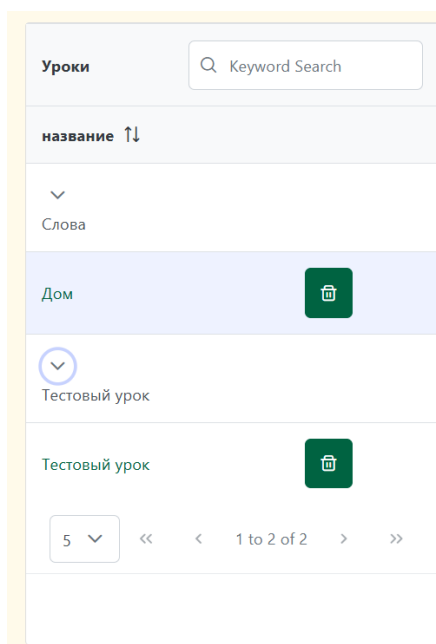


Рисунок 5.45 – Добавленный урок

Урок добавился в список созданных уроков.

### 5.30 Добавление задания

Функция Администратора которая расположена в разделе «Уроки». Для добавления формы необходимо выбрать урок из списка, и заполнить форму добавления задания. На рисунке 5.46 изображена заполненная форма для добавления задания.

Рисунок 5.46 – Форма для добавления заданий

Заполнить форму необходимо предложением на испанском и его перевод на русский. После нажать на кнопку «Добавить». На рисунке 5.47 изображено добавленное задание.

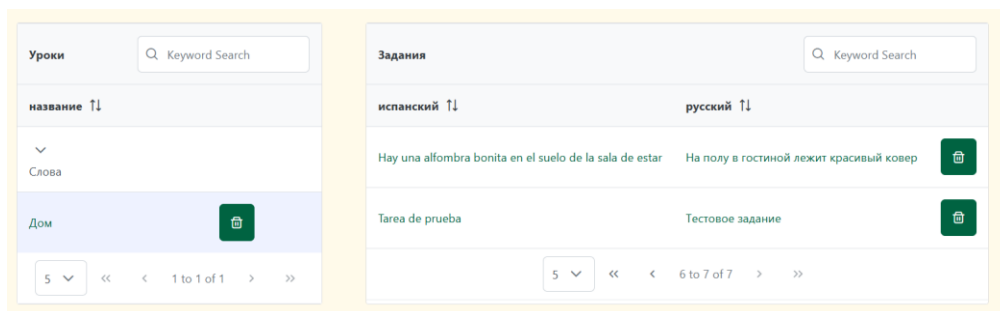


Рисунок 5.47 – Добавленное задание

Задание добавилось в список заданий урока.

### 5.31 Изменение задания

Функция Администратора которая расположена в разделе «Уроки». Для изменения задания необходимо выбрать нужное задание, после этого предложения задания появятся в форме ниже. На рисунке 5.48 изображена форма для добавления и изменения задания после нажатия на существующее задание.

Рисунок 5.48 – Форма для изменения и добавления задания

После ввода изменений необходимо нажать кнопку «Изменить». На рисунке 5.49 изображено измененное задание и уведомление об его успешном изменении.

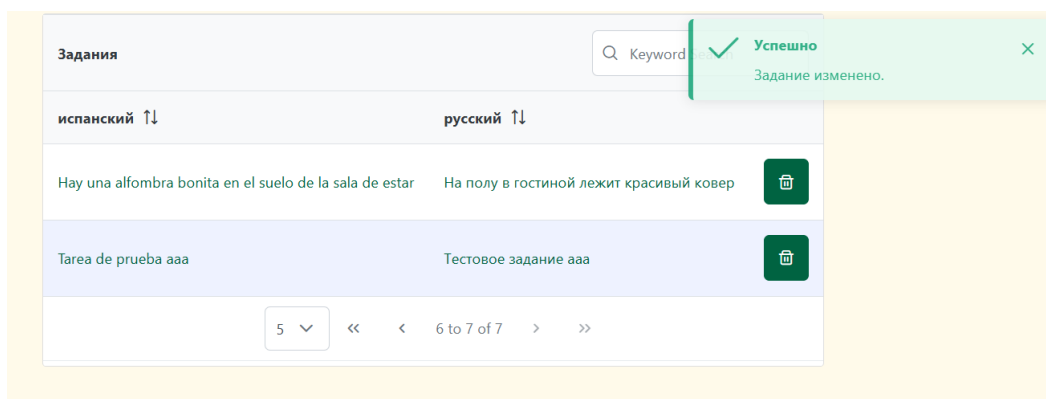


Рисунок 5.49 – Уведомление о успешном изменении задания

После успешного изменения слово изменится в списке заданий и появится уведомление об успешном изменении.

### 5.32 Удаление задания

Функция Администратора которая расположена в разделе «Уроки». Для удаления необходимо выбрать урок и нажать на кнопку с изображением корзины рядом с заданием. На рисунке 5.50 изображена кнопка удаления задания в момент наведения пользователя на нее.

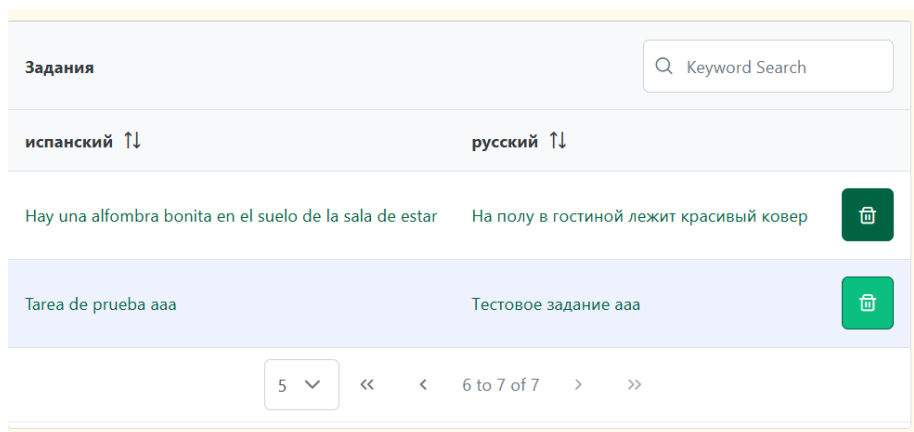


Рисунок 5.50 – Кнопка удаления задания

После нажатия на кнопку удаления задание пропадает из списка заданий. На рисунке 5.51 изображен список заданий после удаления.

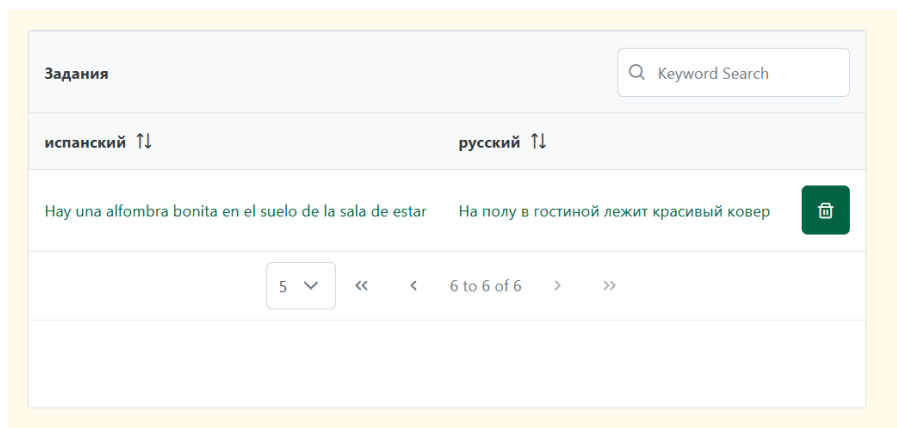


Рисунок 5.51 – Список заданий после удаления задания

Интерфейс отображает актуальный список заданий с внесёнными изменениями.

### 5.34 Просмотр заданий

Функция Администратора и Пользователя которая расположена в разделе «Уроки». Для нее нужно нажать на урок и отобразится список с заданиями, изображенный на рисунке 5.22

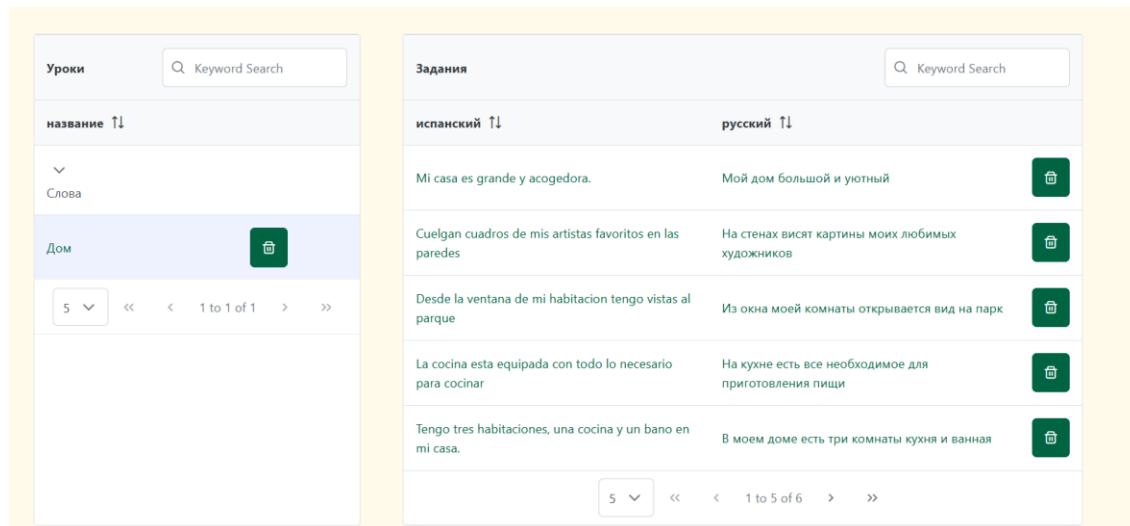


Рисунок 5.52 – Просмотр заданий

Будет выводиться ограниченный список заданий, который можно пролистывать.



## Заключение

В ходе данного курсового проекта было реализовано веб-приложение для изучения испанского языка. Основные результаты разработки:

1. Web-приложение поддерживает 3 роли: Гость, Пользователь, Администратор.
2. В ходе рассмотрения аналогов разрабатываемого приложения были выявлены основные функции и элементы
3. Реализовано 35 функций.
4. База данных включает 14 таблиц.
5. Использована REST-архитектура, обеспечивающая удобное взаимодействие клиента и сервера через HTTP. Основные компоненты: Модели данных: описывают таблицы базы данных и их связи, разработаны с помощью PostgreSQL. Поддержка двусторонней связи через WebSocket для обновлений в реальном времени.
6. Было разработано 39 тестов, которые покрыли 95% функционала веб-приложения.
7. В проекте написано 9332 строки кода, включая серверную и клиентскую части.

На основе полученных результатов работы web-приложения можно заключить, что цель проекта достигнута, а все требования технического задания были полностью выполнены.

### Список использованных источников

1. node [Электронный ресурс]. – Режим доступа: <https://nodejs.org/> – Дата доступа: 12.12.2024.
2. Главная страница сайта Duolingo [Электронный ресурс]. – Режим доступа: <https://www.duolingo.com/> – Дата доступа: 12.12.2024.
3. Главная страница сайта Poliglot16 [Электронный ресурс]. – Режим доступа: <https://poliglot16.ru/> – Дата доступа: 12.12.2024.
4. Главная страница сайта Puzzle English [Электронный ресурс]. – Режим доступа: <https://puzzle-english.com/> – Дата доступа: 12.12.2024.
5. nginx [Электронный ресурс]. – Режим доступа: <https://nginx.org/> – Дата доступа: 12.12.2024.
6. PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> – Дата доступа: 13.12.2024.
7. HTTP [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/HTTP> – Дата доступа: 14.10.2024.
8. HTTPS [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#https> – Дата доступа: 14.10.2024.
9. TCP [Электронный ресурс]. – Режим доступа: <https://www.rfc-editor.org/rfc/rfc793> – Дата доступа: 14.10.2024.
10. WebSocket [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket> – Дата доступа: 14.10.2024.
11. Nest [Электронный ресурс]. – Режим доступа: <https://nestjs.com/> – Дата доступа: 12.12.2024.
12. TypeScript [Электронный ресурс]. – Режим доступа: <https://www.typescriptlang.org/> – Дата доступа: 12.12.2024.
13. JavaScript [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> – Дата доступа: 12.12.2024.
14. ECMAScript [Электронный ресурс]. – Режим доступа: <https://tc39.es/ecma262/> – Дата доступа: 12.12.2024.
15. Prisma [Электронный ресурс]. – Режим доступа: <https://www.prisma.io/> – Дата доступа: 12.12.2024.
16. @nestjs [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@nestjs/core> – Дата доступа: 12.12.2024.
17. @prisma/client [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@prisma/client> – Дата доступа: 12.12.2024.
18. bcrypt [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/bcrypt> – Дата доступа: 12.12.2024.
19. class-transformer [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/class-transformer> – Дата доступа: 12.12.2024.
20. class-validator [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/class-validator> – Дата доступа: 12.12.2024.

21. cookie [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/cookie> – Дата доступа: 12.12.2024.
22. cookie-parser [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/cookie-parser> – Дата доступа: 12.12.2024.
23. cors [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/cors> – Дата доступа: 12.12.2024.
24. helmet [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/helmet> – Дата доступа: 12.12.2024.
25. https [Электронный ресурс]. – Режим доступа: <https://nodejs.org/api/https.html> – Дата доступа: 12.12.2024.
26. joi [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/joi> – Дата доступа: 12.12.2024.
27. nodemailer [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/nodemailer> – Дата доступа: 12.12.2024.
28. passport [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/passport> – Дата доступа: 12.12.2024.
29. passport-jwt [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/passport-jwt> – Дата доступа: 12.12.2024.
30. reflect-metadata [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/reflect-metadata> – Дата доступа: 12.12.2024.
31. rxjs [Электронный ресурс]. – Режим доступа: <https://rxjs.dev/> – Дата доступа: 12.12.2024.
32. websocket [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/ws> – Дата доступа: 12.12.2024.
33. @fortawesome [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@fortawesome/fontawesome-svg-core> – Дата доступа: 12.12.2024.
34. @fortawesome/react-fontawesome [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@fortawesome/react-fontawesome> – Дата доступа: 12.12.2024.
35. @testing-library/jest-dom [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@testing-library/jest-dom> – Дата доступа: 12.12.2024.
36. @testing-library/react [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@testing-library/react> – Дата доступа: 12.12.2024.
37. axios [Электронный ресурс]. – Режим доступа: <https://axios-http.com/> – Дата доступа: 12.12.2024.
38. bootstrap [Электронный ресурс]. – Режим доступа: <https://getbootstrap.com/> – Дата доступа: 12.12.2024.
39. chart.js [Электронный ресурс]. – Режим доступа: <https://www.chartjs.org/> – Дата доступа: 12.12.2024.
40. date-fns [Электронный ресурс]. – Режим доступа: <https://date-fns.org/> – Дата доступа: 12.12.2024.
41. dotenv [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/dotenv> – Дата доступа: 12.12.2024.

42. js-cookie [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/js-cookie> – Дата доступа: 12.12.2024.
43. jwt-decode [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/jwt-decode> – Дата доступа: 12.12.2024.
44. mobx [Электронный ресурс]. – Режим доступа: <https://mobx.js.org/> – Дата доступа: 12.12.2024.
45. primeicons [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/primeicons> – Дата доступа: 12.12.2024.
46. primereact [Электронный ресурс]. – Режим доступа: <https://www.primefaces.org/primereact/> – Дата доступа: 12.12.2024.
47. react [Электронный ресурс]. – Режим доступа: <https://reactjs.org/> – Дата доступа: 12.12.2024.
48. react-bootstrap [Электронный ресурс]. – Режим доступа: <https://react-bootstrap.github.io/> – Дата доступа: 12.12.2024.
49. react-date-range [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-date-range> – Дата доступа: 12.12.2024.
50. react-dom [Электронный ресурс]. – Режим доступа: <https://reactjs.org/docs/react-dom.html> – Дата доступа: 12.12.2024.
51. react-router-dom [Электронный ресурс]. – Режим доступа: <https://reactrouter.com/> – Дата доступа: 12.12.2024.
52. react-scripts [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-scripts> – Дата доступа: 12.12.2024.
53. rxjs [Электронный ресурс]. – Режим доступа: <https://rxjs.dev/> – Дата доступа: 12.12.2024.
54. socket.io-client [Электронный ресурс]. – Режим доступа: <https://socket.io/> – Дата доступа: 12.12.2024.
55. web-vitals [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/web-vitals> – Дата доступа: 12.12.2024.
56. ws [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/ws> – Дата доступа: 12.12.2024.
57. Modular Layered Architecture [Электронный ресурс]. – Режим доступа: <https://martinfowler.com/eaaCatalog/layers.html> – Дата доступа: 12.12.2024.
58. SMTP (Simple Mail Transfer Protocol) [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc5321> – Дата доступа: 12.12.2024.
59. Deepinfra AI [Электронный ресурс]. – Режим доступа: <https://deepinfra.com/> – Дата доступа: 12.12.2024.

## Приложение А

### Листинг схемы базы данных в ORM Prisma

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id            int           @id @default(autoincrement())
  name          String
  email         String        @unique
  password      String
  roleId        int           @default(1) @map("role_id")
  idEnableLesson int         @default(1) @map("id_enable_lesson")
  role          Role          @relation(fields: [roleId], references: [id], onDelete: Cascade)
  session       Session?
  grammarProgress GrammarProgress[]
  settings      UserSettings?
  dictionary    Dictionary[]
  lexiconProgress LexiconProgress[]
  statistics    Statistics[]

  @@map("users")
}

model UserSettings {
  userId      int           @unique @map("user_id")
  user        User          @relation(fields: [userId], references: [id], onDelete: Cascade)
  countRepeatWordForLearned int @map("count_repeat_word_for_learned")
  countRepeatWordsSimultaneously int @map("count_learning_words_simultaneously")
  learningModeWords LearningMode
  learningModeTasks LearningMode

  @@map("users_settings")
}

model Topic {
  id            int           @id @default(autoincrement())
  name          String        @unique
  lessons       Lesson[]

  @@map("topic")
}

```

```

model Lesson {
    id            int            @id @default(autoincrement())
    name          String
    theory        String
    topicId       int            @map("topic_id")
    topic         Topic          @relation(fields: [topicId], refer-
ences: [id], onDelete: Cascade)
    tasks         Task[]

    @@map("lessons")
}

model Task {
    id            int            @id @default(autoincrement())
    spanishSentence String      @map("spanish_sentence")
    russianSentence String      @map("russian_sentence")
    lessonId      int            @map("lesson_id")
    lesson        Lesson        @relation(fields: [lessonId],
references: [id], onDelete: Cascade)
    grammarProgress GrammarProgress[]

    @@unique([spanishSentence, russianSentence, lessonId])
    @@map("tasks")
}

model GrammarProgress {
    id            int            @id @default(autoincrement())
    userId        int            @map("user_id")
    user          User           @relation(fields: [userId], references: [id], onDelete:
Cascade)
    taskId        int            @map("task_id")
    task          Task           @relation(fields: [taskId], references: [id], onDelete:
Cascade)

    @@unique([userId, taskId])
    @@map("grammar_progresses")
}

model Dictionary {
    id            int            @id @default(autoincrement())
    name          String
    description    String?
    creatorId     int            @map("creator_id")
    user          User           @relation(fields: [creatorId],
references: [id], onDelete: Cascade)
    dictionaryToWord DictionaryToWord[]
    words         Word[]

    @@map("dictionaries")
}

model Word {

```

```

    id                int                @id @default(autoincrement())
    spanishSpelling   String              @map("spanish_spelling")
    transcription     String
    russianSpelling   String              @map("russian_spelling")
    description       String?
    lexiconProgress   LexiconProgress[]
    dictionaryToWord  DictionaryToWord[]
    dictionaries      Dictionary[]

    @@unique([spanishSpelling, russianSpelling])
    @@map("words")
}

model DictionaryToWord {
    id                int                @id @default(autoincrement())
    dictionaryId      int
    wordId            int
    dictionary         Dictionary @relation(fields: [dictionaryId], refer-
ences: [id], onDelete: Cascade)
    word              Word              @relation(fields: [wordId], references:
[id], onDelete: Cascade)

    @@unique([dictionaryId, wordId])
    @@map("dictionary_to_word")
}

model LexiconProgress {
    id                int                @id @default(autoincrement())
    progressCount     int                @map("progress_count")
    isLearned         Boolean            @map("is_learned")
    userId            int                @map("user_id")
    user              User              @relation(fields: [userId], references: [id],
onDelete: Cascade)
    wordId            int                @map("word_id")
    word              Word              @relation(fields: [wordId], references: [id],
onDelete: Cascade)

    @@unique([userId, wordId])
    @@map("lexicon_progresses")
}

model Statistics {
    id                int                @id @default(autoincrement())
    date              DateTime
    words             int?
    tasks             int?
    quizPoints        Float?            @map("quiz_points")
    userId            int                @map("user_id")
    user              User              @relation(fields: [userId], references: [id],
onDelete: Cascade)

    @@unique([date, userId])
    @@map("statistics")
}

```

```

}

model Session {
    userId      int      @unique @map("user_id")
    user        User     @relation(fields: [userId], references: [id],
onDelete: Cascade)
    refreshToken String @map("refresh_token")

    @@map("sessions")
}

model Role {
    id          int          @id @default(autoincrement())
    title       String
    permissions Permission[]
    users       User[]

    @@map("roles")
}

model Permission {
    id          int          @id @default(autoincrement())
    descriptor  String
    context     String?
    method      Method
    roleId      int          @map("role_id")
    role        Role        @relation(fields: [roleId], references: [id],
onDelete: Cascade)

    @@map("permissions")
}

enum Method {
    ALL
    GET
    POST
    DELETE
    PUT
    PATCH
}

enum LearningMode {
    TRANSLATE_FROM_spanish
    TRANSLATE_FROM_RUSSIAN
    COMBINED
}

```



## Приложение Б

### Листинг файла package.json серверной части приложения

```
{
  "name": "spanish-api",
  "version": "1.0.0",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r
ts-node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^9.0.0",
    "@nestjs/config": "^2.2.0",
    "@nestjs/core": "^9.0.0",
    "@nestjs/jwt": "^10.0.2",
    "@nestjs/mapped-types": "*",
    "@nestjs/passport": "^9.0.3",
    "@nestjs/platform-express": "^9.0.0",
    "@nestjs/platform-socket.io": "^9.4.0",
    "@nestjs/swagger": "^6.1.4",
    "@nestjs/websockets": "^9.4.0",
    "@prisma/client": "^4.11.0",
    "bcrypt": "^5.1.0",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.0",
    "cookie": "^0.5.0",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "https": "^1.0.0",
    "joi": "^17.9.0",
    "nodemailer": "^6.7.8",
    "passport": "^0.6.0",
    "passport-jwt": "^4.0.1",
    "passport-local": "^1.0.0",
    "reflect-metadata": "^0.1.13",
    "rxjs": "^7.8.1",
    "websocket": "^1.0.34"
  }
}
```

```

    },
    "devDependencies": {
      "@nestjs/cli": "^9.0.0",
      "@nestjs/schematics": "^9.0.0",
      "@nestjs/testing": "^9.0.0",
      "@types/bcrypt": "^5.0.0",
      "@types/cookie-parser": "^1.4.3",
      "@types/express": "^4.17.13",
      "@types/jest": "29.2.4",
      "@types/node": "^18.11.18",
      "@types/passport-jwt": "^3.0.8",
      "@types/passport-local": "^1.0.35",
      "@types/supertest": "^2.0.11",
      "@types/ws": "^8.5.4",
      "@typescript-eslint/eslint-plugin": "^5.0.0",
      "@typescript-eslint/parser": "^5.0.0",
      "eslint": "^8.0.1",
      "eslint-config-prettier": "^8.3.0",
      "eslint-plugin-prettier": "^4.0.0",
      "jest": "29.3.1",
      "prettier": "^2.3.2",
      "prisma": "^4.12.0",
      "source-map-support": "^0.5.20",
      "supertest": "^6.1.3",
      "ts-jest": "29.0.3",
      "ts-loader": "^9.2.3",
      "ts-node": "^10.9.1",
      "tsconfig-paths": "4.1.1",
      "typescript": "^4.9.5"
    },
    "jest": {
      "moduleFileExtensions": [
        "js",
        "json",
        "ts"
      ],
      "rootDir": "src",
      "testRegex": ".*\\.spec\\.ts$",
      "transform": {
        "^.+\\.?(t|j)s$": "ts-jest"
      },
      "collectCoverageFrom": [
        "**/*.?(t|j)s"
      ],
      "coverageDirectory": "../coverage",
      "testEnvironment": "node"
    },
    "prisma": {
      "seed": "ts-node prisma/seed.ts"
    }
  }
}

```

## Приложение В

### Листинг файла package.json клиентской части приложения

```
{
  "name": "spanish-front",
  "proxy": "https://localhost:5000/",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^6.3.0",
    "@fortawesome/free-regular-svg-icons": "^6.3.0",
    "@fortawesome/free-solid-svg-icons": "^6.3.0",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.1.1",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.3",
    "bootstrap": "^5.2.3",
    "chart.js": "^4.3.0",
    "date-fns": "^2.29.3",
    "dotenv": "^16.0.3",
    "js-cookie": "^3.0.1",
    "jwt-decode": "^3.1.2",
    "mobx": "^6.0.5",
    "mobx-react-lite": "^3.1.7",
    "primeicons": "^6.0.1",
    "primereact": "^9.2.3",
    "react": "^18.0.0",
    "react-bootstrap": "^2.7.4",
    "react-date-range": "^1.4.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.10.0",
    "react-scripts": "5.0.1",
    "rxjs": "^7.8.1",
    "socket.io-client": "^4.6.1",
    "web-vitals": "^2.1.4",
    "ws": "^8.13.0"
  },
  "scripts": {
    "start": "set HTTPS=true& set\nSSL_CERT_FILE=C:\\\\My\\\\OpenSSL\\\\CW.crt& set\nSSL_KEY_FILE=C:\\\\My\\\\OpenSSL\\\\CW.key& react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
```

```
"production": [  
  ">0.2%",  
  "not dead",  
  "not op_mini all"  
],  
"development": [  
  "last 1 chrome version",  
  "last 1 firefox version",  
  "last 1 safari version"  
]  
}  
}
```

## Приложение Г

### Код контроллера dictionary

```

import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
  Req,
  Res,
} from '@nestjs/common';
import { ApiBearerAuth, ApiOkResponse, ApiTags } from '@nestjs/swagger';
import { swaggerType } from 'src/helpers/swagger/utils';
import { JwtAuthGuard } from '../auth/guard/jwt-auth.guard';
import RequestWithUser from '../auth/interface/request-with-user.interface';
import { DictionaryService } from '../dictionary.service';
import { CreateDictionaryDto } from '../dto/create-dictionary.dto';
import { UpdateDictionaryDto } from '../dto/update-dictionary.dto';
import { DictionaryReviewResponse } from '../response/dictionary-review.response';
import { DictionaryResponse } from '../response/dictionary.response';
import * as fs from 'fs';
import { Response as ExpressResponse } from 'express';

@ApiTags('dictionary')
@Controller('dictionary')
export class DictionaryController {
  constructor(private readonly dictionaryService: DictionaryService) {}

  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Post()
  public createDictionary(
    @Req() req: RequestWithUser,
    @Body() createDictionaryDto: CreateDictionaryDto,
  ): Promise<DictionaryResponse> {
    return this.dictionaryService.createDictionary(
      req.user.id,
      createDictionaryDto,
    );
  }

  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Get('export/:id')

```

```

    public async export(@Param('id') id: string, @Res() res: ExpressRe-
sponse) {
        const fileName = 'dictionary_' + Date.now() + '.json';
        const fileContent = await this.dictionaryService.exportDiction-
ary(+id);

        fs.writeFileSync(fileName, JSON.stringify(fileContent), 'utf-8');

        res.setHeader('Content-Type', 'application/json');
        res.setHeader('Content-Disposition', `attachment; file-
name=${fileName}`);

        const fileStream = fs.createReadStream(fileName);
        fileStream.pipe(res);

        fileStream.on('end', () => {
            fs.unlinkSync(fileName);
        });
    }

    @ApiBearerAuth()
    @ApiResponse(swaggerType(DictionaryResponse))
    @UseGuards(JwtAuthGuard)
    @Get('admin')
    public getAdminDictionaries(): Promise<DictionaryResponse[]> {
        return this.dictionaryService.getAdminDictionaries();
    }

    @ApiBearerAuth()
    @ApiResponse(swaggerType(DictionaryResponse))
    @UseGuards(JwtAuthGuard)
    @Get('learn')
    public getDictionariesLearn(
        @Req() req: RequestWithUser,
    ): Promise<DictionaryReviewResponse[]> {
        return this.dictionaryService.getDictionariesLearn(+req.user.id);
    }

    @ApiBearerAuth()
    @ApiResponse(swaggerType(DictionaryResponse))
    @UseGuards(JwtAuthGuard)
    @Get('review')
    public getDictionariesReview(
        @Req() req: RequestWithUser,
    ): Promise<DictionaryReviewResponse[]> {
        return this.dictionaryService.getDictionariesRe-
view(+req.user.id);
    }

    @ApiBearerAuth()
    @ApiResponse(swaggerType(DictionaryResponse))
    @UseGuards(JwtAuthGuard)
    @Get('user')

```

```

public getUserDictionaries(
  @Req() req: RequestWithUser,
): Promise<DictionaryResponse[]> {
  return this.dictionaryService.getUserDictionaries(+req.user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('/:id')
public remove(
  @Param('id') id: string,
  @Req() req: RequestWithUser,
): Promise<void> {
  return this.dictionaryService.removeDictionary(+id,
+req.user.id);
}

@ApiBearerAuth()
@ApiOkResponse(swaggerType(DictionaryResponse))
@UseGuards(JwtAuthGuard)
@Patch('/:id')
public updateDictionary(
  @Param('id') id: string,
  @Req() req: RequestWithUser,
  @Body() updateDictionaryDto: UpdateDictionaryDto,
): Promise<DictionaryResponse> {
  return this.dictionaryService.updateDictionary(
    +id,
    +req.user.id,
    updateDictionaryDto,
  );
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post('import/:id')
public async importDictionary(
  @Param('id') id: string,
  @Req() req: RequestWithUser,
  @Body() body: { property: any[] }, // Указываем ожидаемую
структуру
): Promise<void> {
  const wordData = body.property; // Извлекаем массив из property
  await this.dictionaryService.importDictionary(+id, wordData);
}
}

```

## Приложение Д

### Код сервиса dictionary

```
import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { Dictionary } from '@prisma/client';
import { DictionaryRepository } from '../dictionary.repository';
import { CreateDictionaryDto } from '../dto/create-dictionary.dto';
import { UpdateDictionaryDto } from '../dto/update-dictionary.dto';
import { DictionaryReviewResponse } from '../response/dictionary-review.response';
import { DictionaryResponse } from '../response/dictionary.response';
import { WordForExportResponse } from '../word/response/word-for-export.response';

@Injectable()
export class DictionaryService {
  constructor(private readonly dictionaryRepository: DictionaryRepository) {}

  public async checkDictionaryOwner(
    dictionaryId: number,
    userId: number,
  ): Promise<void> {
    const dictionary: Dictionary =
      await this.dictionaryRepository.getDictionaryById(dictionaryId);

    if (!dictionary) {
      throw new NotFoundException('Dictionary not found');
    }

    if (dictionary.creatorId !== userId) {
```



```

        throw new BadRequestException('Not owner try to update dictionary');
    }
}

public async importDictionary(
    dictionaryId: number,
    wordData: WordForExportResponse[],
): Promise<void> {
    const dictionary = await this.dictionaryRepository.getDictionaryById(
        dictionaryId,
    );

    if (!dictionary) {
        throw new NotFoundException('Dictionary not found');
    }

    await this.dictionaryRepository.importWords(dictionaryId, wordData);
}

public async createDictionary(
    creatorId: number,
    createDictionaryDto: CreateDictionaryDto,
): Promise<DictionaryResponse> {
    return await this.dictionaryRepository.createDictionary(
        creatorId,
        createDictionaryDto,
    );
}

public async exportDictionary(
    dictionaryId: number,
): Promise<WordForExportResponse[]> {

```

```

        return await this.dictionaryRepository.exportDictionaryById(dictionaryId);
    }

    public async getAdminDictionaries(): Promise<DictionaryResponse[]> {
        return await this.dictionaryRepository.getAdminDictionaries();
    }

    public async getDictionariesLearn(
        id: number,
    ): Promise<DictionaryReviewResponse[]> {
        const data = await this.dictionaryRepository.getDictionariesLearn(id);

        data.forEach((dictionary) => {
            dictionary.words = dictionary.words.filter(
                (word) =>
                    word.lexiconProgress[0] === undefined ||
                    word.lexiconProgress[0].isLearned === false,
            );
        });

        return data;
    }

    public async getDictionariesReview(
        id: number,
    ): Promise<DictionaryReviewResponse[]> {
        return await this.dictionaryRepository.getDictionariesReview(id);
    }

    public async getUserDictionaries(id: number): Promise<DictionaryResponse[]> {
        return await this.dictionaryRepository.getUserDictionaries(id);
    }

```

```
}

public async removeDictionary(
    dictionaryId: number,
    userId: number,
): Promise<void> {
    await this.checkDictionaryOwner(dictionaryId, userId);
    await this.dictionaryRepository.removeDictionaryById(dictionaryId);
}

public async updateDictionary(
    dictionaryId: number,
    userId: number,
    updateDictionaryDto: UpdateDictionaryDto,
): Promise<DictionaryResponse> {
    await this.checkDictionaryOwner(dictionaryId, userId);

    return await this.dictionaryRepository.updateDictionaryById(
        dictionaryId,
        updateDictionaryDto,
    );
}
}
```

## Приложение Е

### Код репозитория dictionary

```

import { Injectable } from '@nestjs/common';

import { DatabaseService } from 'src/database/database.service';
import { CreateDictionaryDto } from '../dto/create-dictionary.dto';
import { UpdateDictionaryDto } from '../dto/update-dictionary.dto';
import { DictionaryResponse } from '../response/dictionary.response';
import { WordForExportResponse } from '../word/response/word-for-export.response';

@Injectable()
export class DictionaryRepository {
  private dictionariesReviewSelect = {
    id: true,
    name: true,
    description: true,
    creatorId: true,
    words: {
      select: {
        id: true,
        spanishSpelling: true,
        transcription: true,
        russianSpelling: true,
        description: true,
        lexiconProgress: {
          select: {
            progressCount: true,
            isLearned: true,
          },
        },
      },
    },
  },
};

private fullDictionarySelect = {
  id: true,
  name: true,
  description: true,
  creatorId: true,
  words: {
    select: {
      id: true,
      spanishSpelling: true,
      transcription: true,
      russianSpelling: true,
      description: true,
    },
  },
};

constructor(private readonly db: DatabaseService) {}

```

```

public async createDictionary(
    creatorId: number,
    createDictionaryDto: CreateDictionaryDto,
): Promise<DictionaryResponse> {
    return await this.db.dictionary.create({
        data: {
            creatorId,
            ...createDictionaryDto,
        },
        select: this.fullDictionarySelect,
    });
}

public async exportDictionaryById(
    id: number,
): Promise<WordForExportResponse[]> {
    const dictionary = await this.db.dictionary.findMany({
        where: {
            id,
        },
        select: {
            words: {
                select: {
                    spanishSpelling: true,
                    transcription: true,
                    russianSpelling: true,
                    description: true,
                },
            },
        },
    });

    return dictionary[0].words;
}

public async getAdminDictionaries(): Promise<DictionaryResponse[]>
{
    return await this.db.dictionary.findMany({
        where: {
            user: {
                roleId: 2,
            },
        },
        select: this.fullDictionarySelect,
    });
}

public async getDictionariesLearn(creatorId: number): Promise<any[]> {
    return await this.db.dictionary.findMany({
        where: {
            OR: [{ creatorId }, { user: { roleId: 2 } }],
        },
    });
}

```

```

select: {
  id: true,
  name: true,
  description: true,
  creatorId: true,
  words: {
    select: {
      id: true,
      spanishSpelling: true,
      transcription: true,
      russianSpelling: true,
      description: true,
      lexiconProgress: {
        where: {
          userId: creatorId,
        },
        select: {
          id: true,
          progressCount: true,
          isLearned: true,
        },
        take: 1,
      },
    },
  },
},
});
}

public async getDictionariesReview(creatorId: number): Promise<any[]> {
  return await this.db.dictionary.findMany({
    where: {
      OR: [{ creatorId }, { user: { roleId: 2 } }],
    },
    select: {
      id: true,
      name: true,
      description: true,
      creatorId: true,
      words: {
        select: {
          id: true,
          spanishSpelling: true,
          transcription: true,
          russianSpelling: true,
          description: true,
          lexiconProgress: {
            where: {
              userId: creatorId,
            },
            select: {
              progressCount: true,

```

```

        isLearned: true,
    },
    take: 1,
},
},
},
},
});
}

public async getDictionaryById(id: number): Promise<DictionaryResponse> {
    return await this.db.dictionary.findUnique({
        where: {
            id,
        },
        select: this.fullDictionarySelect,
    });
}

public async getUserDictionaries(
    creatorId: number,
): Promise<DictionaryResponse[]> {
    return await this.db.dictionary.findMany({
        where: {
            creatorId,
        },
        select: this.fullDictionarySelect,
    });
}

public async removeDictionaryById(id: number): Promise<void> {
    await this.db.dictionary.delete({
        where: {
            id,
        },
    });
}

public async updateDictionaryById(
    id: number,
    updateDictionaryDto: UpdateDictionaryDto,
): Promise<DictionaryResponse> {
    return await this.db.dictionary.update({
        where: {
            id,
        },
        data: {
            ...updateDictionaryDto,
        },
        select: this.fullDictionarySelect,
    });
}

```

```

public async importWords(
  dictionaryId: number,
  wordData: WordForExportResponse[],
): Promise<void> {
  try {
    console.log('Data to insert:', wordData);

    // Шаг 1: Создать записи в таблице Word
    const createdWords = await this.db.word.createMany({
      data: wordData.map((word) => ({
        spanishSpelling: word.spanishSpelling,
        transcription: word.transcription,
        russianSpelling: word.russianSpelling,
        description: word.description,
      })),
      skipDuplicates: true,
    });

    console.log('Words created:', createdWords);

    // Шаг 2: Получить слова из базы данных (с их ID)
    const words = await this.db.word.findMany({
      where: {
        OR: wordData.map((word) => ({
          spanishSpelling: word.spanishSpelling,
          russianSpelling: word.russianSpelling,
        })),
      },
    });

    // Шаг 3: Создать записи в таблице DictionaryToWord
    const dictionaryToWordData = words.map((word) => ({
      dictionaryId: dictionaryId,
      wordId: word.id,
    }));

    await this.db.dictionaryToWord.createMany({
      data: dictionaryToWordData,
      skipDuplicates: true,
    });

    console.log('DictionaryToWord relationships created.');
```

```

  } catch (error) {
    console.error('Error during importWords:', error);
    throw error;
  }
}

```



## Приложение Ж

### Код страницы PersonalCabinet

```

import { useContext, useEffect, useRef, useState } from "react";
import { Button } from "primereact/button";
import { Toast } from "primereact/toast";
import { Slider } from "primereact/slider";
import { Dropdown } from "primereact/dropdown";

//theme
import "primereact/resources/themes/lara-light-indigo/theme.css";

//core
import "primereact/resources/primereact.min.css";

//icons
import "primeicons/primeicons.css";

import { observer } from "mobx-react-lite";
import Input from "../components/Input";
import { REGEXES } from "../utils/regexes";

import { changePassword, updateUserSettings } from "../api-requests/user-api";

import "../styles/personal-cabinet.css";
import { Context } from "..";
import { LearningMode } from "../utils/learn-settings";

const PersonalCabinet = observer(() => {
  const toast = useRef(null);
  const { userSettings } = useContext(Context);
  const { user } = useContext(Context);

  useEffect(() => {
    setLearningSettings({
      countRepeatWordForLearned: userSettings.getCountRepeatWordFor-
Learned(),
      countRepeatWordsSimultaneously:
        userSettings.getCountRepeatWordsSimultaneously(),
      learningModeWords: userSettings.getLearningModeWords(),
      learningModeTasks: userSettings.getLearningModeTasks(),
    });
  }, [userSettings]);

  const showSuccess = (message) => {
    toast.current.show({
      severity: "success",
      summary: "Успешно",
      detail: message,
      life: 3000,
    });
  };
};

```

```

const showError = (message) => {
  toast.current.show({
    severity: "error",
    summary: "Ошибка",
    detail: message,
    life: 3000,
  });
};

const [password, setPassword] = useState({
  oldPassword: "",
  newPassword: "",
  repitNewPassword: "",
});
const [isValidPassword, setIsValidPassword] = useState(null);
const [isValidPasswordRepit, setIsValidPasswordRepit] = use-
  State(null);

const changePasswordHandler = ({ id, value }) => {
  setPassword((prev) => ({ ...prev, [id]: value }));
  console.log(user.getRoleId());

  if (id === "newPassword") {
    setIsValidPassword(REGEXES.PASSWORD_REGEX.test(value));
    setIsValidPasswordRepit(password.repitNewPassword === value);
  }

  if (id === "repitNewPassword") {
    setIsValidPasswordRepit(password.newPassword === value);
  }
};

const changePasswordRequest = async () => {
  try {
    await changePassword(password.oldPassword, password.newPass-
word);
    setPassword({
      oldPassword: "",
      newPassword: "",
      repitNewPassword: "",
    });
    showSuccess("Пароль изменён");
  } catch (error) {
    showError("Введён неверный текущий пароль");
  }
};

const [learningSettings, setLearningSettings] = useState({
  countRepeatWordForLearned: userSettings.getCountRepeatWordFor-
Learned(),
  countRepeatWordsSimultaneously:
    userSettings.getCountRepeatWordsSimultaneously(),

```

```

    learningModeWords: userSettings.getLearningModeWords(),
    learningModeTasks: userSettings.getLearningModeTasks(),
  });

  const changeSettingsHandler = (id, value) => {
    if (learningSettings[id] !== value) {
      setLearningSettings((prev) => ({ ...prev, [id]: value }));
    }
  };

  const learningModes = [
    {
      name: "Перевод с испанского",
      value: LearningMode.TRANSLATE_FROM_SPANISH,
    },
    { name: "Перевод с русского", value: LearningMode.TRANSLATE_FROM_RUSSIAN },
    { name: "Комбинированный вариант", value: LearningMode.COMBINED },
  ],
];

const updateUserSettingsRequest = async () => {
  try {
    await updateUserSettings(learningSettings);
    userSettings.setUserSettings(learningSettings);
    showSuccess("Настройки обучения обновлены.");
  } catch (error) {
    showError();
  }
};

return (
  <div className="container">
    <Toast ref={toast} />
    <div className="account-settings">
      <h4 className="">Обновление данных аккаунта</h4>
      <Input
        id="oldPassword"
        value={password.oldPassword}
        labelText="Текущий пароль"
        isValidValue={true}
        inputType="password"
        setDataHandler={changePasswordHandler}
      />
      <Input
        id="newPassword"
        value={password.newPassword}
        labelText="Новый пароль"
        isValidValue={isValidPassword}
        inputType="password"
        setDataHandler={changePasswordHandler}
        errorMessage="От 4 до 16 символов"
      />
    </div>
  </div>
);

```

```

<Input
  id="repitNewPassword"
  value={password.repitNewPassword}
  labelText="Повторите новый пароль"
  isValidValue={isValidPasswordRepit}
  inputType="password"
  setDataHandler={changePasswordHandler}
  errorMessage="Пароли не совпадают"
/>
<Button
  className="change-password-button"
  label="Сменить пароль"
  onClick={changePasswordRequest}
  disabled={!isValidPassword || !isValidPasswordRepit}
/>
</div>
<div className="learning-settings">
  <h4>Настройки обучения</h4>
  <div className="learning-settings__block">
    <label className="learning-settings__label">
      Количество правильных ответов, следующих подряд, после
которого
      слово считается выученным и исключается из списка, но
остаётся в
      словаре
    </label>
    <Slider
      className="learning-settings__input"
      value={learningSettings.countRepeatWordForLearned}
      min={2}
      max={10}
      onChange={ (e) =>
        changeSettingsHandler("countRepeatWordForLearned",
e.value)
      }
    </Slider>
    <label>{learningSettings.countRepeatWordForLearned}</label>
  </div>
  <div className="learning-settings__block">
    <label className="learning-settings__label">
      Количество слов изучаемых одновременно
    </label>
    <Slider
      className="learning-settings__input"
      value={learningSettings.countRepeatWordsSimultaneously}
      min={10}
      max={50}
      onChange={ (e) =>
        changeSettingsHandler("countRepeatWordsSimultaneously",
e.value)
      }
    </Slider>
  </div>

```

```

        <label>{learningSettings.countRepeatWordsSimultane-
ously}</label>
    </div>
    <div className="learning-settings__block">
        <label className="learning-settings__label">
            Способ изучения лексики
        </label>
        <Dropdown
            value={learningSettings.learningModeWords}
            onChange={(e) =>
                changeSettingsHandler("learningModeWords", e.value)
            }
            options={learningModes}
            optionLabel="name"
            placeholder="Выберите вариант"
            className="w-full"
        />
    </div>
    <div className="learning-settings__block">
        <label className="learning-settings__label">
            Способ изучения грамматики
        </label>
        <Dropdown
            value={learningSettings.learningModeTasks}
            onChange={(e) =>
                changeSettingsHandler("learningModeTasks", e.value)
            }
            options={learningModes}
            optionLabel="name"
            placeholder="Выберите вариант"
            className="w-full"
        />
    </div>
    <Button
        className="update-user-settings-button"
        label="Сохранить"
        onClick={updateUserSettingsRequest}
    />
</div>
</div>
);
});

export default PersonalCabinet;

```