

## РЕФЕРАТ

ВКР бакалавра «Адаптивный сервис в помощь изучающему иностранный (английский) язык».

Пояснительная записка: 102 страницы, 38 рисунков, 7 таблиц, 10 источников, 1 приложение.

Ключевые слова: иностранный язык, английский язык, лексика, WEB-приложение, информационная система, программный продукт.

Целью работы является разработка WEB-приложения для помощи в изучении английской лексики с использованием мнемонического подхода.

Разрабатываемое приложение призвано помочь пользователю со средними и высокими знаниями английского языка быстрее изучать лексику английского языка. Для этого в приложении используются сформированные у пользователя ассоциации с местом встречи данной лексики в оригинальных источниках: книгах, фильмах, журналах и статьях. Кроме того, при изучении лексики приложение предоставляет определения изучаемой лексики и примеры ее использования в контексте изучаемого языка.

## СОДЕРЖАНИЕ

СОДЕРЖАНИЕ .....	5
ВВЕДЕНИЕ.....	7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, СУЩЕСТВУЮЩИХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, СИСТЕМ ИЛИ МЕТОДОВ И АЛГОРИТМОВ, КОТОРЫЕ РЕШАЮТ АНАЛОГИЧНЫЕ ЗАДАЧИ.....	10
1.1 Анализ предметной области .....	10
1.2 Обзор существующих аналогов.....	15
1.3 Формулирование требований к разрабатываемому приложению ...	26
Выводы по разделу 1 .....	27
2 СИСТЕМОТЕХНИЧЕСКИЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «АДАПТИВНАЯ СИСТЕМА В ПОМОЩЬ ИЗУЧАЮЩЕМУ ИНОСТРАННЫЙ (АНГЛИЙСКИЙ) ЯЗЫК» .	28
2.1 Построение диаграмм потоков данных (DFD) в проектируемой системе .....	28
2.2 Построение функциональных и информационных моделей IDEF0- IDEF1 проектируемой системы .....	30
2.3 Разработка алгоритма функционирования приложения .....	34
2.4 Разработка структуры данных .....	39
Выводы к разделу 2.....	43
3 РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ «АДАПТИВНАЯ СИСТЕМА В ПОМОЩЬ ИЗУЧАЮЩЕМУ ИНОСТРАННЫЙ (АНГЛИЙСКИЙ) ЯЗЫК» И ЕЕ ПОДСИСТЕМ .....	44
3.1 Выбор и обоснование информационного обеспечения ИС.....	44
3.2 Выбор и обоснование языка программирования и инструментальных средств для разработки.....	51
3.3 Общее описание архитектуры системы.....	57
3.4 Программная реализация информационной системы.....	58
Выводы к разделу 3.....	68

ЗАКЛЮЧЕНИЕ .....	69
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ .....	70
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	72
ПРИЛОЖЕНИЕ А .....	74
Программный код приложения .....	74

## ВВЕДЕНИЕ

Английский язык является одним из самых используемых в мире. Согласно данным крупнейшего в мире каталога языков за 2021 год [1] английский язык занимает 1 место в топ 200 самых используемых языков мира. Общее число владеющих английским языком оценивается в 1,35 млрд человек, число носителей языка – 379 млн человек (3 место в мире после китайского и испанского). Оценивается, что не менее 80% научных публикаций, 75% всей коммуникации на международном уровне, 80% информации, хранящейся на компьютерах, 90% контента в интернете в мире находятся на английском языке. Из этого следует, что знание английского языка необходимо большинству людей. Кроме того, тот факт, что английский, не будучи первым по количеству коренных носителей языка, является самым используемым языком, говорит о том, что в современном мире существует большой спрос на изучение этого языка.

Обучение любому естественному языку производится как минимум на двух уровнях: грамматики и лексики. На начальном этапе изучения иностранного языка в одинаковой степени возникает необходимость как в изучении его лексики, так и грамматики. Но при продвижении вглубь грамматики языка, необходимость в дальнейшем изучении грамматики падает, так как степень понимания прочитанного или услышанного начинает больше зависеть от словарного запаса потребителя. И чем дальше человек проходит в процессе изучения иностранного языка, тем больше перевешивает необходимость изучения лексики. Данная работа сконцентрирована именно на эффективном изучении английской лексики.

Чтобы соответствовать быстрому развитию науки и техники необходимо уметь запоминать большое количество информации и уметь оперировать ей. Процесс изучения лексики мало чем отличается от запоминания большого количества информации любого другого вида.

Любая информация, на которой человек концентрируется первый раз, так или иначе попадает сначала в краткосрочную память. Чтобы перенести эту информацию в долгосрочное хранилище необходимо правильное ее запоминание.

Самым распространенным способом запоминания информации является механическое запоминание. Смысл механического запоминания состоит в запоминании информации в той форме, в которой она воспринимается. Основной проблемой данного способа является быстрое забывание этой информации, ведь для ее перехода в долгосрочную память необходимо формирование прочных связей в мозге человека. Для этого необходимо повторять процесс изучения материала или использовать более совершенные методы запоминания информации.

Мнемотехника – совокупность приемов и техник запоминания информации. Основной ее идеей является составление ассоциаций между запоминаемой информацией и информацией, которая легко вспоминается, с помощью определенных приемов. Большинство существующих приложений ориентируются только на быстрое запоминание информации. При этом смысловая составляющая изучаемой лексики (варианты использования, словосочетания и т.д.) отходит на второй план. В итоге эффективно запоминаются именно ассоциации, которые выстроены для лексических единиц, но возникают проблемы при использовании изученной лексики на практике (при переводе или разговоре). В данной работе смысловая составляющая изучаемой лексики не останется без внимания.

**Цель и задачи работы.** Целью этой дипломной работы является разработка WEB-ориентированного приложения для помощи в изучении английской лексики с использованием мнемонического подхода. Приложение вместо заранее подготовленной базы слов для заучивания будет использовать лексику, встреченную пользователем при потреблении источников на английском языке. Для этого пользователю нужно будет только ввести каждое

встреченное ему слово (словосочетание, выражение). Затем приложение будет предлагать упражнения на запоминание введенной ранее лексики.

Для достижения поставленной цели необходимо:

- проанализировать схожие по функционалу продукты, сравнить их с рассмотренной задумкой;
- проработать сценарии взаимодействия пользователя с приложением;
- осуществить проектирование структуры базы данных (БД) для хранения состояния приложения;
- осуществить проектирование архитектуры приложения для удовлетворения рассмотренных сценариев;
- осуществить разработку программных модулей приложения.

**Предмет и проект исследования.** Объектом исследования данной работы является изучение лексики английского языка. Предметом исследования является WEB-ориентированная информационная система изучения лексики английского языка.

## **1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, СУЩЕСТВУЮЩИХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, СИСТЕМ ИЛИ МЕТОДОВ И АЛГОРИТМОВ, КОТОРЫЕ РЕШАЮТ АНАЛОГИЧНЫЕ ЗАДАЧИ**

В данном разделе рассматривается процесс анализа заданий (упражнений) над изучаемой лексикой для расширения словарного запаса иностранного (английского) языка в мобильных и WEB-приложениях на предмет возможного включения их в разрабатываемое приложение. Далее, будут проанализированы существующие мобильные и WEB-приложения для расширения словарного запаса иностранного языка и составлены функциональные и нефункциональные требования к разрабатываемому приложению, которое будет реализовано по окончании данной работы.

### **1.1 Анализ предметной области**

#### **1.1.1 Анализ особенностей человеческой памяти в области запоминания однотипной информации**

Большинство недавних исследований в области изучения английского языка с использованием электронных средств (электронных курсов, мобильных и WEB-приложений) положительно влияет на обучение в целом. Но отмечается, что такие средства при изучении грамматики иностранного языка и развитии способности общего использования иностранного языка должны использоваться как дополнение при обучении с педагогом. В случае же изучения лексики иностранного языка после достижения достаточного уровня владения грамматикой самостоятельное изучение лексики не только допускается, но и становится нормой. Поэтому процесс формирования требований будет построен таким образом, чтобы пользователю было возможно пользоваться приложением без необходимости обращения к наставнику.

Изучение иностранной лексики состоит в запоминании однотипной информации. Процесс запоминания неотрывно связан со структурой и принципами работы человеческой памяти. Самой распространенной моделью памяти на данный момент является модель Аткинсона-Шиффрина (также «многоэтажная модель памяти»), предложенная Ричардом Аткинсоном и Ричардом Шиффрином в 1968 году [2]. Согласно этой модели, выделяют три уровня памяти:

- сенсорная – в которой в течение небольшого времени хранится сенсорная информация (менее чем 0,5 секунды для визуальной информации и 2 секунд для звуковой), поступающая из сенсорной системы, возникающая при воздействии стимулов на органы чувств;

- кратковременная — структура, в которую из сенсорной памяти под воздействием внимания заносится и хранится в течение менее 20 секунд небольшой объём информации о 5 - 7 объектах;

- долгосрочная – структура большого объёма, которая может хранить воспоминания вплоть до смерти.

Информация, воспринимаемая в момент рассмотрения очередной лексической единицы и ее перевода, если для нее не сформировано в памяти никаких связей (ассоциаций или сопутствующей информации), на короткое время остается в краткосрочной памяти и вскоре забывается. Об этом свидетельствуют, в том числе, исследования немецкого психолога Эббингауза от 1885 года [3]. В ходе исследований было установлено, что до 60% заученной информации, которая для изучающего не имеет смысла и не вызывает никаких ассоциаций, забывается уже в течение первого часа. Далее процесс забывания идёт медленно, и через 6 дней в памяти остаётся около 20% от общего числа первоначально выученной информации, столько же остаётся в памяти и через месяц. Выводы, которые можно сделать на основании данной кривой в том, что для эффективного запоминания необходимо повторение заученного материала. Кривая Эббингауза изображена на рисунке 1.1.



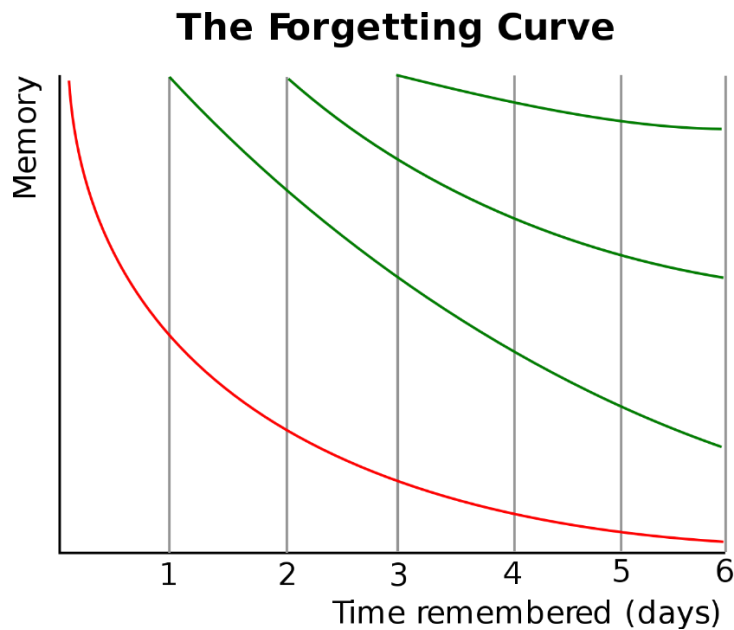


Рисунок 1.1 – Кривая забывания Эббингауза

Для перехода усваиваемой лексики в долгосрочную память необходимо, во-первых, сформировать в памяти соответствующие новой информации связи, и, во-вторых, периодически повторять усваиваемую лексику.

Для реализации системы повторений для запоминания изучаемой лексики необходимо составить список простых лексических упражнений, которые не требуют ни методических указаний, ни контроля со стороны преподавателя.

### **1.1.2 Упражнения, используемые для запоминания английской лексики**

Изучение иностранных языков не является новой задачей. С тех пор, когда английский вследствие развития английских колоний по всему миру, начал считаться международным, существовала необходимость в его изучении. Опыт изучения иностранных языков миллионами человек и преподавателей по всему миру, особенно с середины XX века, аккумулировался в множество материалов, рекомендаций и приложений. Рассмотрим доступную информацию в поиске простых упражнений для

наполнения словарного запаса, которые не требуют методических указаний и контроля преподавателя. Результат приведен далее:

1. Определить, правилен ли предложенный перевод. Пользователю показывается лексическая единица и ее перевод. Перевод может быть верен или неверен. Пользователю необходимо дать лишь ответ, верен ли перевод.

2. Выбор правильного перевода слова среди предложенного списка переводов. Пользователю предоставляется лексическая единица, которую необходимо перевести и от 3 до 6 вариантов перевода, среди которых только один правильный. Неправильные варианты переводов желательно формировать из множества переводов лексики, которые были внесены в систему примерно в то же время (или из того же словаря, если лексика сгруппирована по смыслу), что и предложенная к переводу лексическая единица.

3. Выбор правильной лексической единицы по ее переводу. То же самое, что и выбор правильного перевода по предложенному списку переводов, только наоборот.

4. Ввод лексической единицы по ее переводу. Пользователю показан перевод лексической единицы, необходимо ввести с помощью клавиатуры данную лексическую единицу. Возможны ошибки в написании слов. Правильность ответа определяется в зависимости от количества допущенных ошибок.

5. Ввод лексической единицы по ее звучанию. Пользователю предлагается прослушать звучание лексической единицы, продиктованное диктором (или синтезированное с помощью «Text to Speech» сервисов). После этого предлагается ввести услышанную лексическую единицу с клавиатуры. При этом возможны ошибки. Правильность ответа определяется в зависимости от количества допущенных ошибок.

6. Выбор правильного перевода лексической единицы по ее звучанию. Пользователю предлагается прослушать звучание лексической единицы, продиктованное диктором (или синтезированное с помощью «Text to Speech»

сервисов). После этого предоставляется от 3 до 6 вариантов перевода, среди которых необходимо выбрать правильный. Неправильные варианты переводов желательно формировать из множества переводов лексики, которые были внесены в систему примерно в то же время (или из того же словаря, если лексика сгруппирована по смыслу), что и прослушанная лексическая единица.

7. «Собери пару». Пространство страницы делится на блоки, каждый из которых содержит либо лексическую единицу, либо перевод. Количество переводов и лексических единиц равно, каждой лексической единице соответствует только один перевод. Количество участвующих лексических единиц – не более 6 (не более 12 блоков всего). Блоки лексических единиц и переводов выделены разными цветами. Блоки расставляются в случайном порядке. Пользователю необходимо составить соответствие между каждой лексической единицей и ее переводом. Правильность ответа определяется количеством правильных соответствий.

8. «Мемория». Пространство страницы делится на блоки, каждый из которых содержит либо лексическую единицу, либо перевод. Количество переводов и лексических единиц равно, каждой лексической единице соответствует только один перевод. Количество участвующих лексических единиц – не более 5 (не более 10 блоков всего). Блоки лексических единиц и переводов выделены разными цветами. Блоки расставляются либо в 2 столбца, либо в 2 строки таким образом, что все лексические единицы находятся в одной строке или столбце, а переводы – в другой строке или столбце. Тексты лексических единиц или переводов доступны только первые 5 или 10 секунд, после чего становятся невидимы. Пользователю необходимо составить соответствие между блоками каждой лексической единицы и блоком ее перевода.

9. «Собери слово». Пользователю предоставляется перевод лексической единицы и список блоков, представляющих каждую букву этой лексической единицы. Блоки, соответствующие буквам лексической единицы расположены в случайном порядке. Пользователю необходимо выбрать блоки

букв в таком порядке, в каком они расположены в лексической единице. Правильность ответа определяется в зависимости от количества допущенных ошибок.

10. Заполнение пропущенных в слове мест. Пользователю представляется лексическая единица, в которой пропущены некоторые буквы. Пропуски пользователю предлагается заполнить с клавиатуры. Ответ считается верным только если правильно заполнены все пропуски.

## **1.2 Обзор существующих аналогов**

Существует несколько программ, которые позиционируют себя как приложения для эффективного и удобного изучения английской лексики. Рассматриваются программы, реализованные не только в виде WEB-сайтов, но и в виде мобильных приложений. Самыми популярными и заслуживающими интереса считаются LinguaLeo, Quizlet, Easy Ten, Duolingo. Кроме того, заслуживают интереса не столь популярные приложения Words и Brainscape. Некоторые из них, например, LinguaLeo, являются приложениями более широкого плана, предоставляющими гораздо больше функций, чем словарь для эффективного изучения лексики: уроки грамматики, прослушивание аудиокниг, специально подготовленные видеоролики на английском языке. Такие функции в обзоре будут опускаться т.к. не являются частью данной работы. Кроме того, в некоторых из приведенных выше приложений представлены функции для работы с другими иностранными языками, которые тоже не будут рассматриваться.

### **1.2.1 Анализ функций приложения LinguaLeo**

Для пополнения словарного запаса в LinguaLeo существует раздел тренировок «словарные». В этом разделе доступны следующие виды тренировок:

– «Брейншторм» – тренировка, в которой вам предлагают список слов, среди которых нужно выбрать те, которые вы не знаете. Тогда они будут добавлены к следующим тренировкам другого вида. Экран тренировки «Брейншторм» изображен на рисунке 1.2.

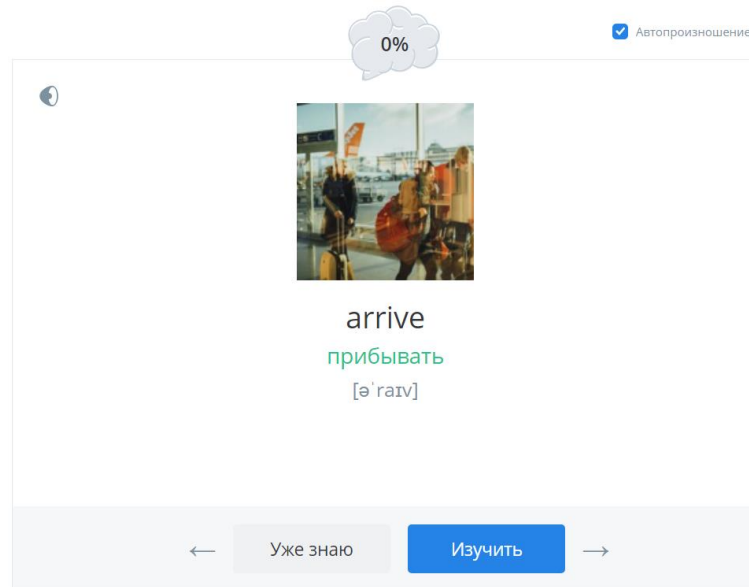


Рисунок 1.2 – Экран тренировки «Брейншторм»

– «Перевод-слово» – тренировка, в которой для данного перевода необходимо выбрать соответствующее ему слово. Экран тренировки «Перевод-слово» изображен на рисунке 1.3.

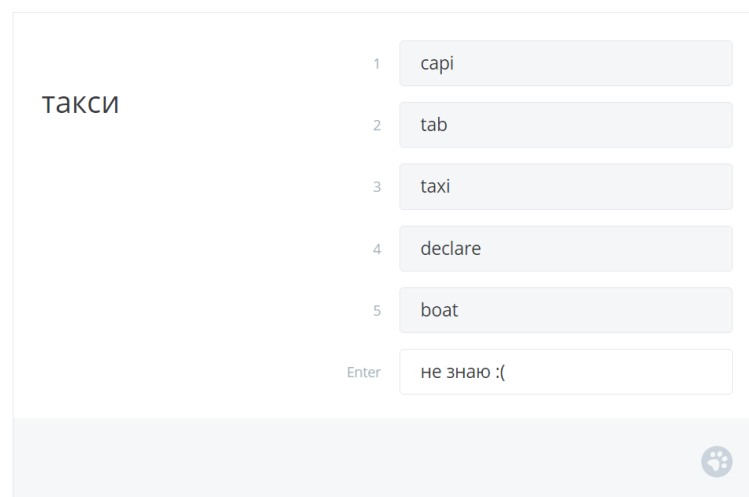


Рисунок 1.3 – Экран тренировки «Перевод-слово»

– «Слово-перевод» – то же самое, что и тренировка «Перевод-слово», только вместо выбора слова по его переводу предлагается выбрать перевод по слову. Экран тренировки «Слово-перевод» выглядит точно так же, как и экран тренировки «Перевод-слово».

– Аудирование – тренировка, в которой предлагается прослушать слово, произносимое диктором и записать его с клавиатуры. Экран аудирования приведен на рисунке 1.4.

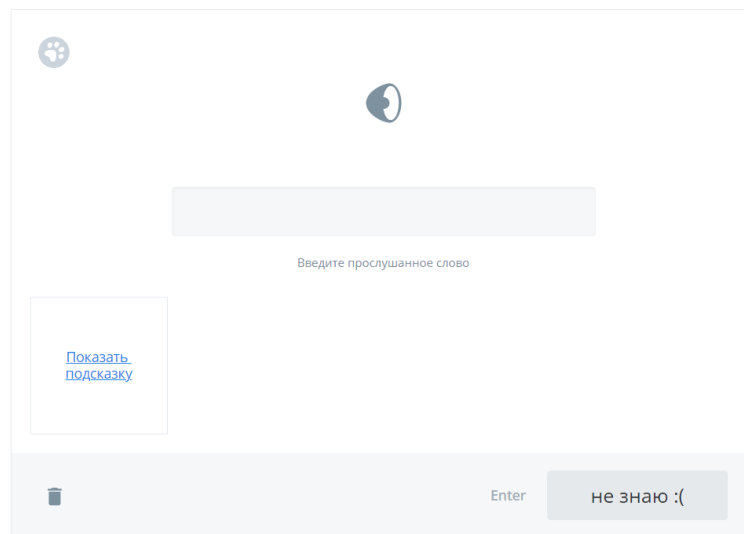


Рисунок 1.4 – Экран аудирования

– «Кроссворд» – тренировка, в которой предлагаются переводы по вертикали и горизонтали и сетка кроссворда, которую пользователю необходимо заполнить. Экран тренировки изображен на рисунке 1.5.

– «Саванна» – тренировка, в которой предлагается выбирать перевод слова на время. Если пользователь не успел выбрать перевод, то он теряет одну жизнь. Число жизней ограничено. Экран тренировки «Саванна» изображен на рисунке 1.6.

Все тренировки проходят над неким тематическим набором слов. Наборы слов заранее predetermined, но есть возможность задать собственный набор. В этом случае нужно ввести только слово, которое хотите

добавить. Все возможности словарных тренировок доступны только по платной подписке. Также доступен и журнал, в котором отмечаются успехи

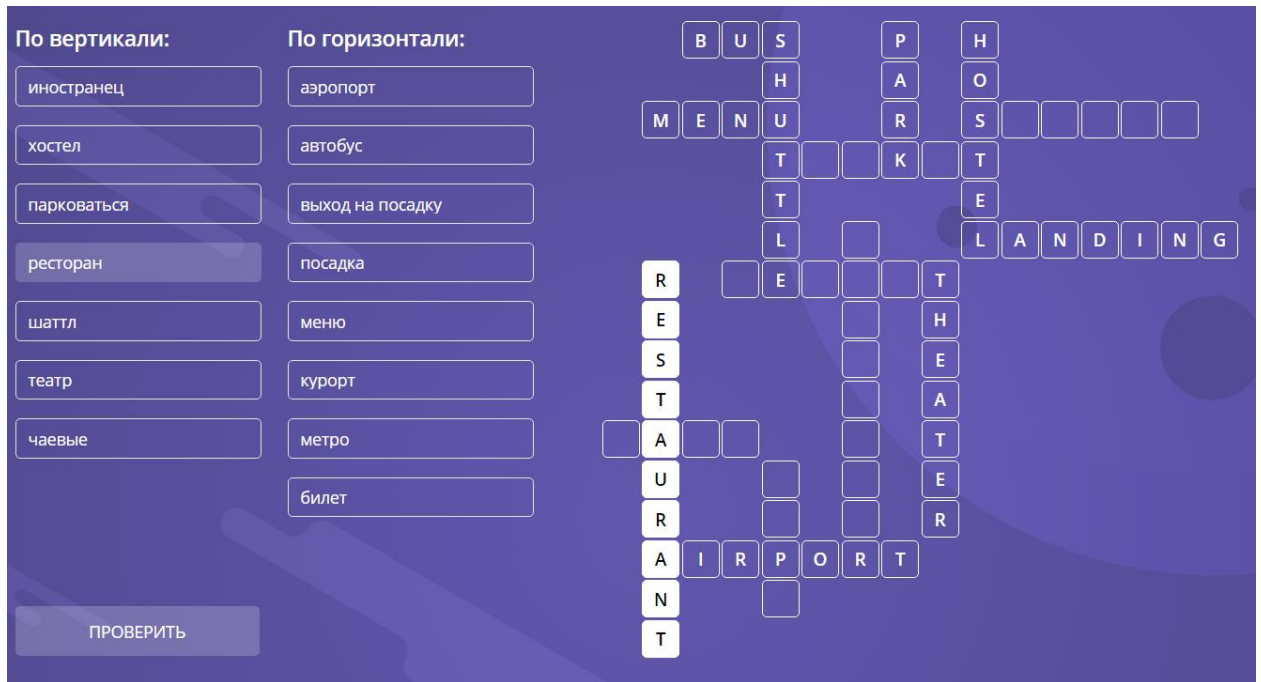


Рисунок 1.5 – Экран тренировки «Кроссворд»

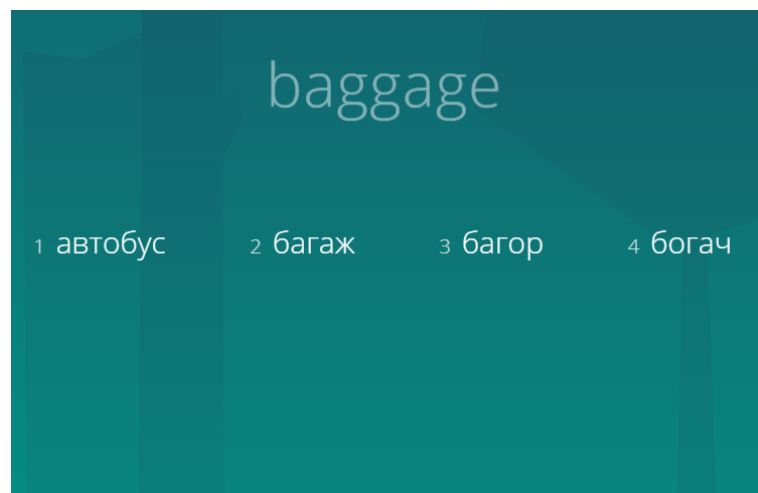


Рисунок 1.6. – Экран тренировки «Саванна»

В целом, приложение LinguaLeo богато функциями расширения словарного запаса: оно содержит упражнения для работы со словарями, при этом словари могут быть созданы самим пользователем. Недостатком является то, что все эти функции доступны только по платной подписке. Кроме того,

приложение позволяет контекстом задавать только одно предложение, что плохо подходит для слов, которые могут использоваться в приложении по-разному.

### 1.2.2 Анализ функций Quizlet

Quizlet – приложение для изучения иностранных слов с использованием флэш-карточек (двусторонних карточек, где с одной стороны записано слово, а с другой – его перевод). Quizlet реализует данный способ изучения иностранных слов в электронном виде. В приложении доступны следующие тренировки:

– «Заучивание» – тренировка, в которой по переводу нужно выбрать из списка слово или наоборот. При этом есть возможность прослушать данный перевод или слово, что сильно помогает при изучении. Экран тренировки изображен на рисунке 1.7.

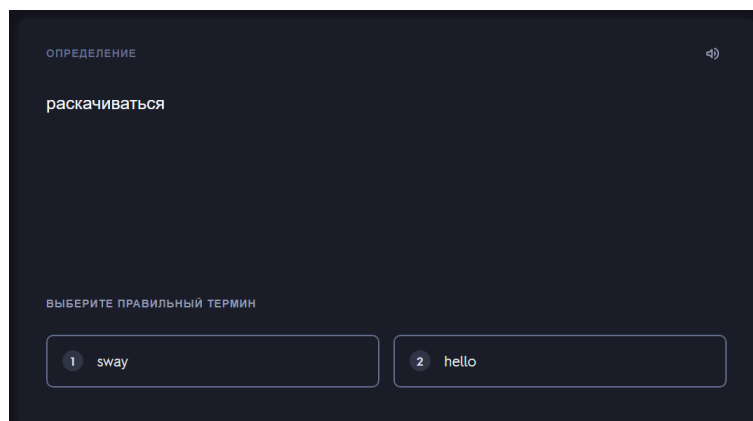


Рисунок 1.7 – Экран тренировки «Заучивание»

– «Письмо» – тренировка, в которой нужно по переводу ввести с клавиатуры слово. Экран тренировки изображен на рисунке 1.8.

– «Правописание» – тренировка, в которой нужно ввести слово по его звучанию. Экран тренировки «Правописание» изображен на рисунке 1.9.



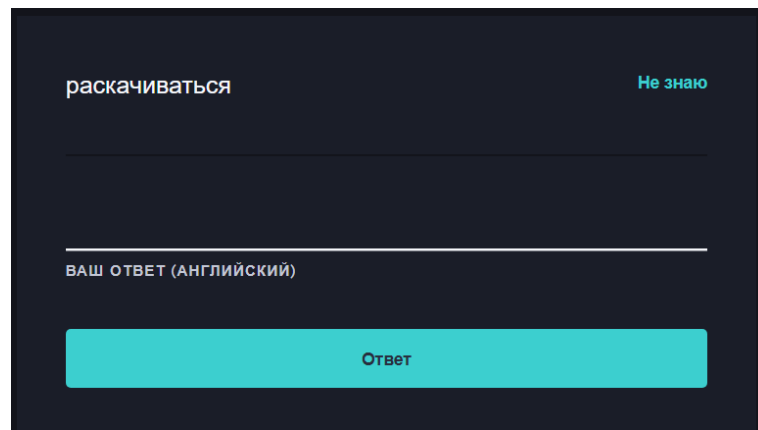


Рисунок 1.8 – Экран тренировки «Письмо»

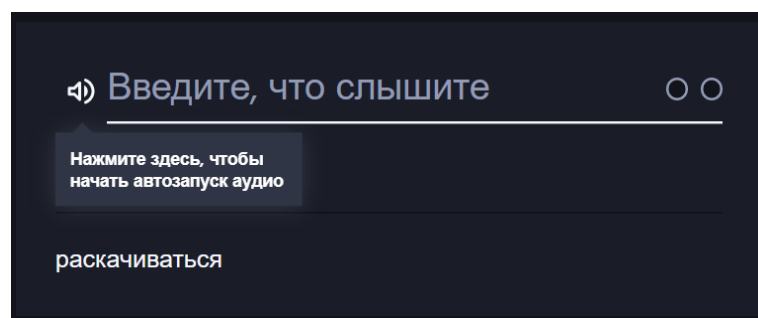


Рисунок 1.9 – Экран тренировки «Правописание»

– «Подбор» – тренировка, в которой слова и переводы размещаются в блоках, которые затем нужно сопоставить между собой. Экран тренировки «Подбор» изображен на рисунке 1.10.

Приложение Quizlet использует те же тренировки, что и приложение LinguaLeo, но рассчитано оно на то, что пользователь будет создавать тренировки сам или получать их от преподавателя т.к. встроенных словарей в нем нет. Некоторые возможности, такие как журнал тренировок и их результатов и другие доступны только по платной подписке. Недостатком приложения Quizlet является то, что вместе с изучаемым слово не идет никакой сопутствующей информации, которая была бы полезна для формирования ассоциаций.



Рисунок 1.10 – Экран тренировки «Подбор»

### 1.2.3 Анализ функций приложения Easy Ten

Easy Ten – мобильное приложение, которое создано для того, чтобы пользователь мог изучать иностранную лексику не напрягаясь. Оно состоит в том, что каждый день пользователю будут предложены 10 слов для изучения с помощью уведомлений в смартфоне. Предложение для изучения является простой флэш-карточкой, а упражнений в приложении не предусмотрено. Таким образом, каждый день изучая новые слова и повторяя старые, можно без особых усилий, по словам авторов, изучать 300 слов в месяц или 3650 слов в год.

На мой взгляд, такой подход является тупиковым в том случае, когда необходимо изучить иностранный язык для реального его использования в жизни так как при простом пролистывании флэш-карточек в памяти не формируются долгосрочные связи для рассматриваемых лексических единиц. Кроме того, не любая лексика, которая была изучена таким образом, в последствии может быть использована из-за полного отсутствия примеров использования и любой другой контекстной информации, а отсутствие пользовательского словаря делает невозможным его использование при изучении источников информации на иностранном языке.

### 1.2.4 Анализ функций приложения Duolingo

Приложение Duolingo является сервисом для общего обучения иностранного языка, включая лексику. Все его функции сосредоточены в интерактивных упражнениях, которые расставлены последовательно по возрастанию уровня владения английским языком. Порядок и содержание этих упражнений жестко зафиксированы, что хорошо подходит только для обучающихся с низким уровнем владения языком. Упражнения чаще всего комбинированы: в них одновременно изучается лексика и грамматика. Кроме упражнений существует журнал изученных слов, который показывает степень овладения каждым словом и позволяет перейти в режим повторения, который будет запускать упражнения, пройденные давно или в которых были допущены ошибки. Типовое упражнение приложения Duolingo изображено на рисунке 1.11.

#### Введите перевод на русский

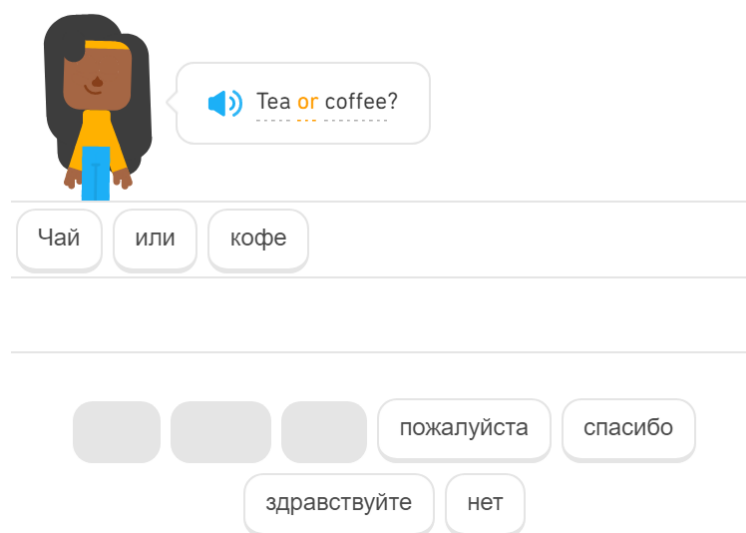


Рисунок 1.11 – Типовой экран упражнения приложения Duolingo

Приложения Duolingo не концентрируется на изучении лексики, но тем не менее, является одним из самых популярных из-за эффективности общего изучения английского языка. Приложение предоставляет большинство функций по платной подписке. Недостатком приложения является полное

отсутствие пользовательского словаря, что делает невозможным его использование при самостоятельном изучении источников информации на иностранном языке.

### 1.2.5 Анализ функций приложения Words

Мобильное приложение Words специализируется на изучении лексики английского языка. В нем доступны те же упражнения, что и в приведенных выше приложениях, со словами из предустановленных наборов или из пользовательских наборов. При этом из 363 уроков бесплатно доступны только 20 первых. Пользовательские словари позволяют задать список переводов слова, но не позволяют задать его контекстную информацию (примеры использования, картинки и так далее). Пользовательские словари также доступны лишь по платной подписке. Приложение отслеживает уровень владения каждым отдельным словом учитывая правильность и частоту ответов на него и используя эту информацию формирует новый список упражнений над лексикой. Типовой экран упражнения приложения Words изображен на рисунке 1.12.

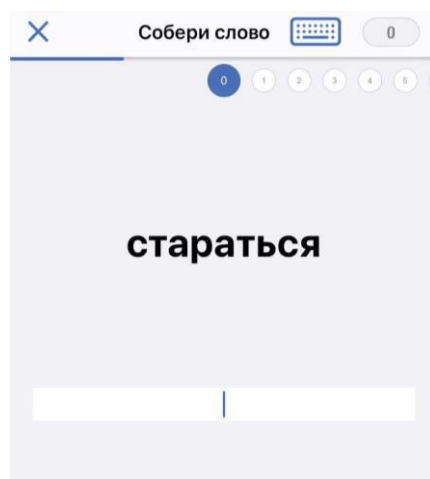


Рисунок 1.12 – Типовой экран упражнения приложения Words

Кроме того, упражнение использует уведомления смартфона для сообщения нового слова для изучения по расписанию.

### **1.2.6 Анализ функций приложения Brainscape**

Приложение Brainscape является универсальной электронной реализацией флэш-карточек и не специализируется на изучении иностранных языков в общем и иностранной лексики в частности. Оно предоставляет платформу для заполнения двух сторон карточки любой информацией, которая необходима пользователю. Это реализовано через HTML редактор карточки при создании перевода. Создатели приложения Brainscape уверяют, что приложение успешно используется в различных сферах человеческой деятельности и в изучении иностранной лексики в том числе.

Для запоминания используются повторяющиеся упражнения следующего вида: пользователю предоставляется одна часть карточки и необходимо про себя ответить, что находится с другой стороны карточки. Пользователю придется оценить себя самостоятельно по пятибалльной шкале и, в зависимости от этого ответа, будет определено, когда в следующий раз включить эту карточку в тренировку. Набор карточек можно создать самостоятельно или использовать существующий. Карточки нужно делить на наборы колод и колоды, что может быть удобно для классификации по иностранным источникам, которые эти карточки покрывают. Кроме того, в приложении доступны множества наборов карточек, доступных для изучения. Недостатком приложения является то, что при использовании его для изучения лексики необходимо вручную искать и вводить переводы и примеры использования, что может занимать много времени. Типовой экран упражнения изображен на рисунках 1.13 (передняя сторона карточки) и 1.14 (задняя сторона).

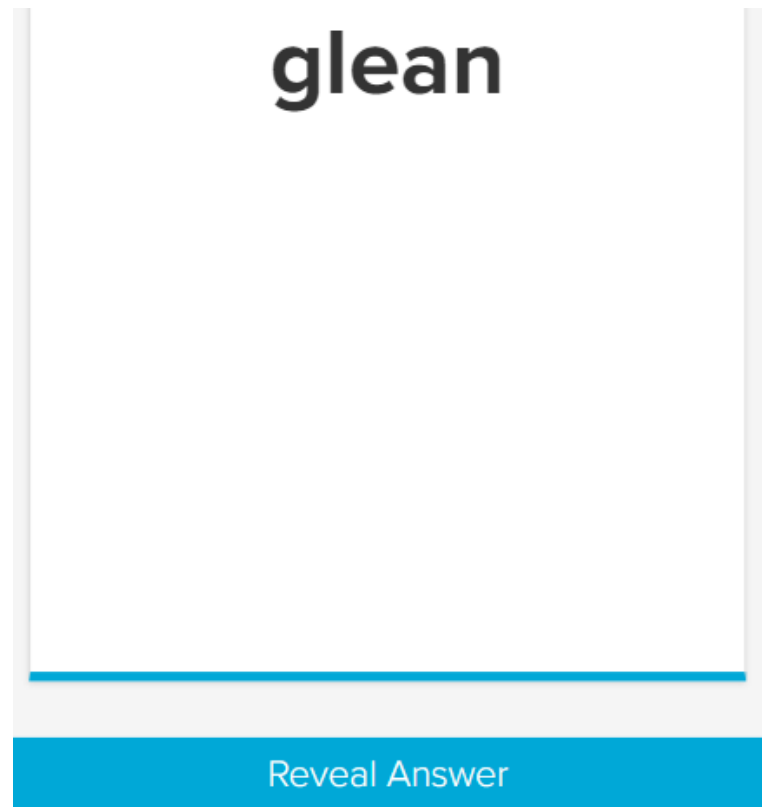


Рисунок 1.13 – Передняя сторона карточки в упражнении Brainscape

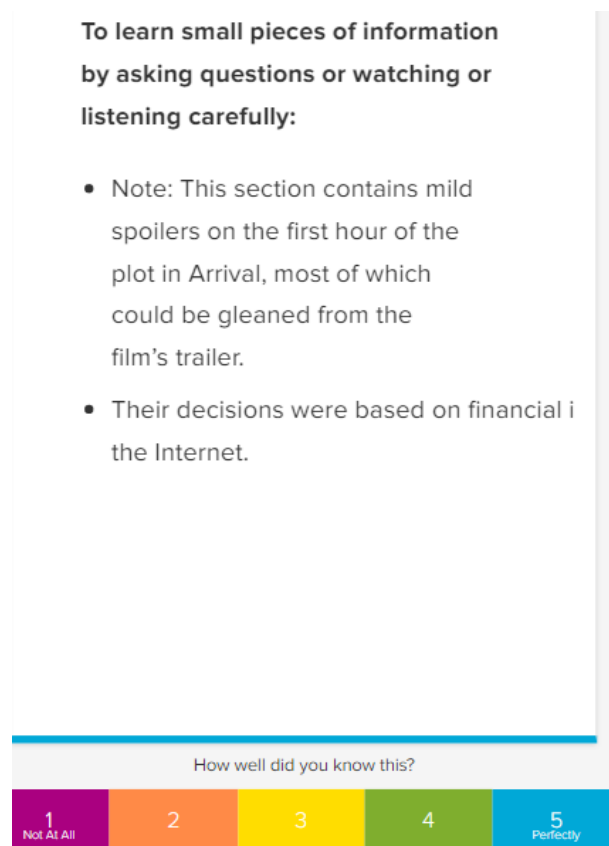


Рисунок 1.14 – Задняя сторона карточки в приложении Brainscape

### 1.3 Формулирование требований к разрабатываемому приложению

Из анализа приложений LinguaLeo, Quizlet, Easy Ten, Words, Brainscape становится понятно, что ни одно из них в полной мере не реализует идею заведения персонального словаря пользователя с возможностью автоматизации создания переводов, дополнения их любой контекстной информацией и создания по ним интерактивных упражнений.

Составим список требований, накладываемых на создаваемое приложение:

- приложение должно быть WEB-ориентированным;
- интерфейс приложения должен быть удобным и адаптивным;
- приложение должно иметь авторизацию для получения возможности создания пользовательских словарей;
- приложение должно позволять создавать наборы словарей, которые используются для группировки словарей;
- приложение должно позволять создавать словари внутри наборов словарей;
- приложение должно позволять создавать переводы внутри словарей (требуется только ввод лексической единицы, по которой должны быть сформированы список переводов и список контекстов использования);
- приложение должно формировать следующие упражнения из подраздела 1.1.1: определить, правилен ли предложенный перевод, выбор правильного перевода слова среди предложенного списка переводов, выбор правильной лексической единицы по ее переводу, ввод лексической единицы по ее переводу.

## **Выводы по разделу 1**

В первом разделе были рассмотрены интерактивные средства, используемые для изучения лексики английского языка, а также мобильные и WEB-приложения, в которых эти средства используются. После их анализа стало понятно, что ни одно из них в полной мере не реализует идею заведения персональных словарей с возможностью автоматизации создания переводов и дополнения их любой необходимой пользователю контекстной (ассоциативной) информацией. Некоторые из них направлены скорее на быстрое запоминание, минуя проблему возможности дальнейшего использования лексики; другие не имеют возможности создания пользовательского словаря и полагаются только на предустановленные; третьи имеют хорошую систему наполнения всевозможной информацией, но в них не интегрирована автоматизация создания переводов. Приложение, реализующее сформулированную идею, находится в разработке.

По результатам анализа приложений-аналогов были сформированы требования к разрабатываемому приложению.



## **2 СИСТЕМОТЕХНИЧЕСКИЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «АДАПТИВНАЯ СИСТЕМА В ПОМОЩЬ ИЗУЧАЮЩЕМУ ИНОСТРАННЫЙ (АНГЛИЙСКИЙ) ЯЗЫК»**

### **2.1 Построение диаграмм потоков данных (DFD) в проектируемой системе**

В системе изучения английской лексики основными процессами является взаимодействие пользователя со словарями, содержащими его лексику. Единственной внешней сущностью для системы является ее пользователь.

Пользователю доступны следующие функции:

1. Создание групп словарей. При создании группы словарей пользователю предлагается ввести имя и описание группы словарей.
2. Создание словарей. При создании словарей пользователю предлагается ввести имя и описание словаря
3. Создание переводов внутри словарей. При создании перевода внутри словаря пользователю необходимо ввести переводимое слово (словосочетание, выражение, предложение), затем выбрать переводы, которые он хотел бы запомнить (либо все предложенные по умолчанию) отредактировать при желании флэш-карточку (по умолчанию генерируется автоматически).
4. Получение списков групп словарей, списков словарей, списков переводов. Это событие предоставляет пользователю возможность просмотреть уже добавленные данные, выбрать их для редактирования.
5. Редактирование перевода и его описания. Предоставляет пользователю возможность отредактировать перевод.
6. Получение списка упражнений и решение этих упражнений. По словарю, списку словарей, выборке словарей или по всем словарям

пользователя можно построить персонифицированный набор упражнений в зависимости от последнего участия слова в упражнениях пользователя, результатов пользователя и т.д.

Внутренних событий не предусмотрено.

Создадим диаграммы потоков данных в соответствии с описанием возможных действий внешних сущностей. Результат изображен на рисунках 2.1 – 2.3.

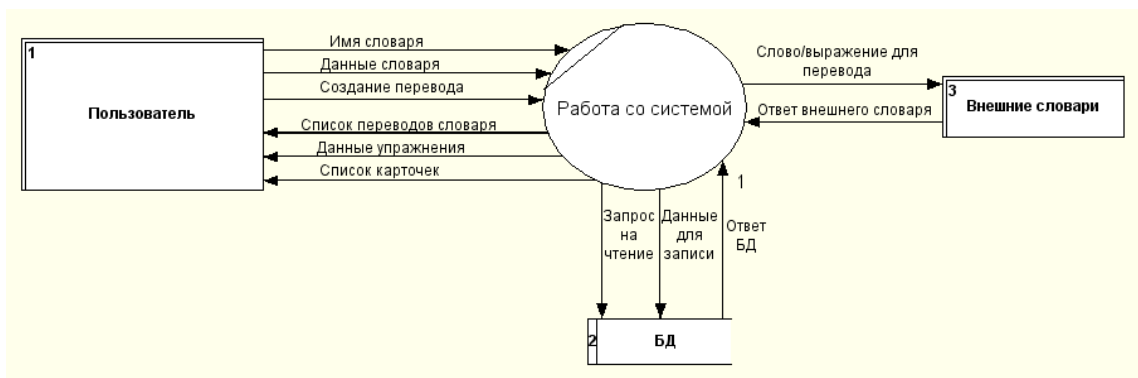


Рисунок 2.1 – DFD-диаграмма основного процесса системы

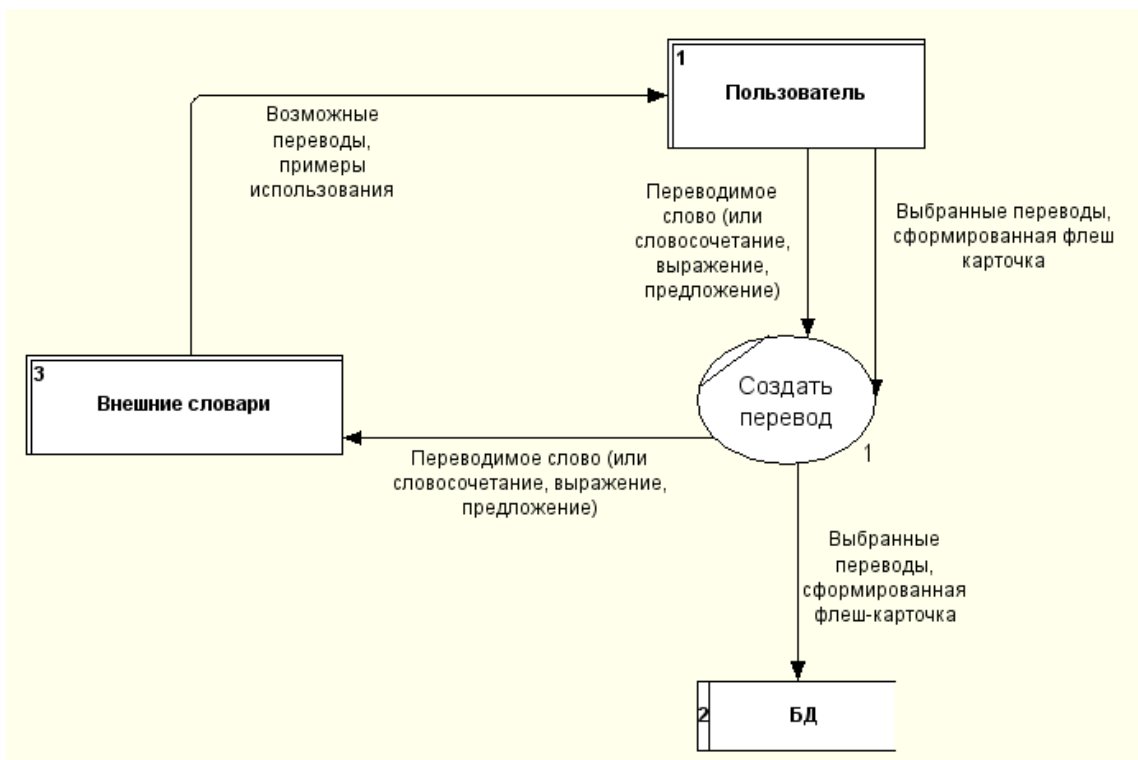


Рисунок 2.2 – DFD-диаграмма процесса создания перевода

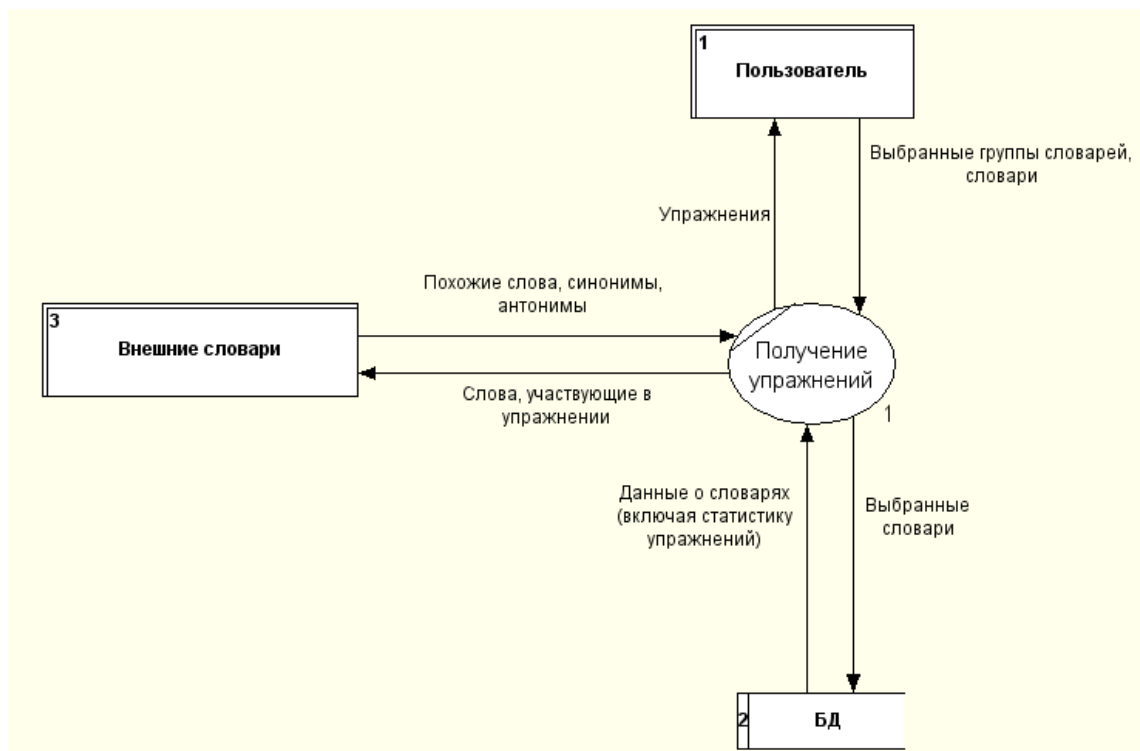


Рисунок 2.3 – DFD-диаграмма процесса получения упражнений

## 2.2 Построение функциональных и информационных моделей IDEF0-IDEF1 проектируемой системы

Для создания контекстной диаграммы процесса взаимодействия пользователя с системой разберем возможные варианты взаимодействий и данные, которые при этих взаимодействиях нужны. Результат представлен в таблице 2.1.

Используя полученные процессы при использовании системы (таблица 2.1) составим контекстную диаграмму основных процессов системы в нотации IDEF0. Результат изображен на рисунке 2.4. Используя контекстную диаграмму, создадим диаграмму дерева узлов системы. Она изображена на рисунке 2.5.

Таблица 2.1 – Детализация основных процессов системы

Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
Создание группы словарей	Название и описание группы словарей	Запрос на создание группы словарей	Административная часть системы	Группа словарей
Создание словаря	Название и описание словаря	Запрос на создание словаря	Административная часть системы	Словарь
Создание перевода	Переводимая лексическая единица, выбранные переводы, составленная контекстная информация	Запрос на создание перевода	Административная часть системы	Перевод
Получение упражнения	Выбранные группы словарей или словари	Запрос на получение упражнений	Механизм формирования упражнений	Подборка упражнений для прохождения

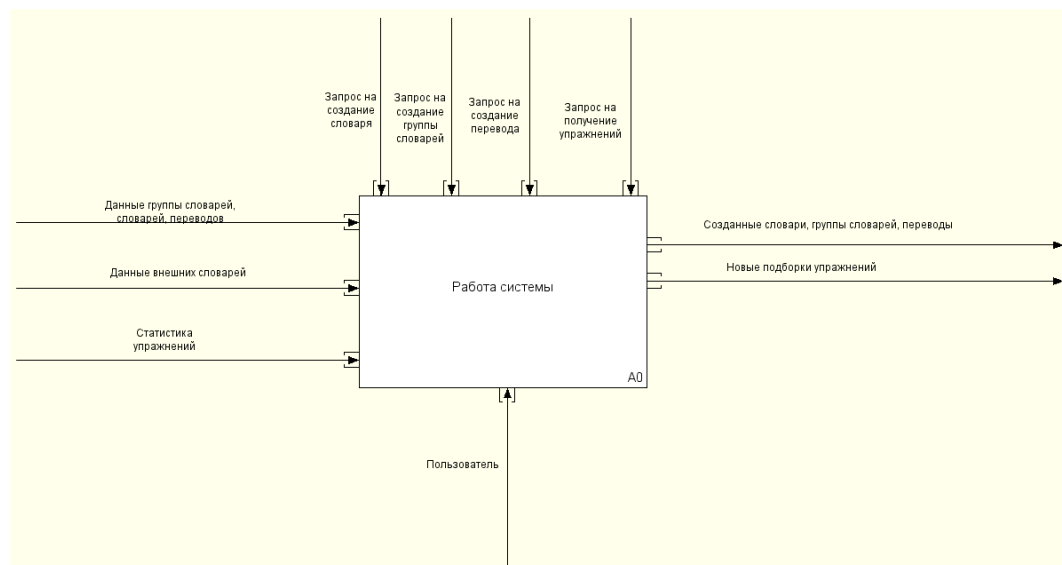


Рисунок 2.4 – Контекстная диаграмма основных процессов системы в нотации IDEF0

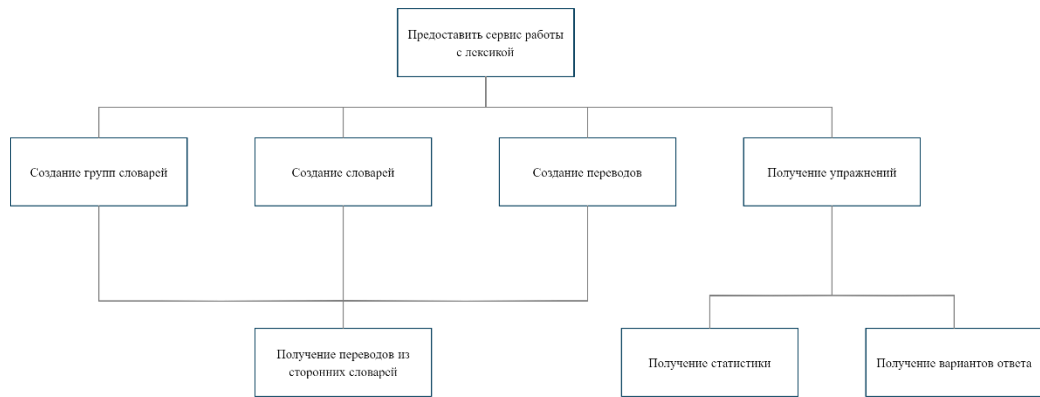


Рисунок 2.5 – Диаграмма дерева узлов системы

Также, используя таблицу 2.1 детализируем основные процессы системы в диаграмме детализации используя те же процессы, что использовались для создания контекстной диаграммы. Результат изображен на рисунке 2.6.

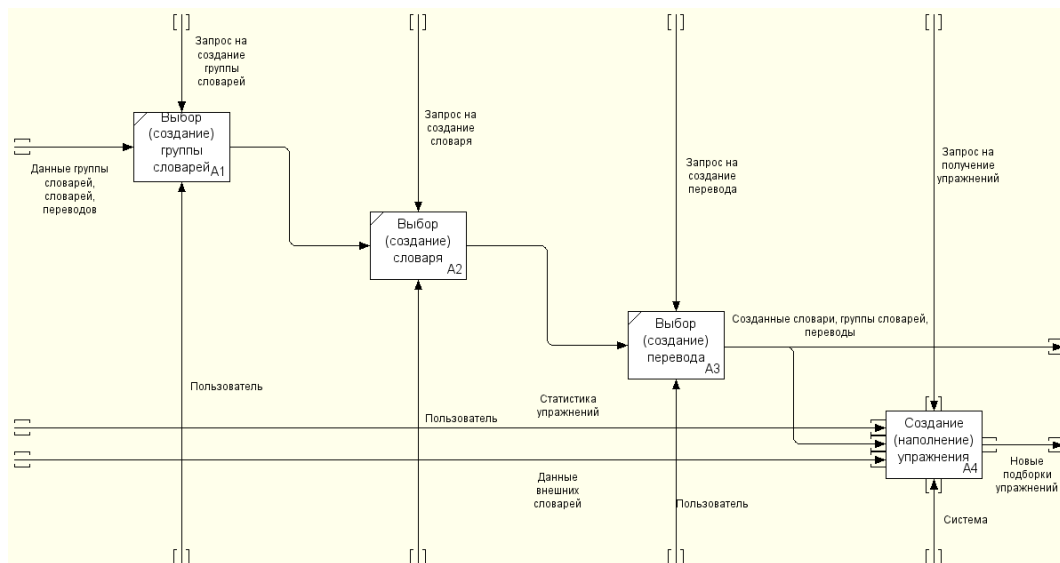


Рисунок 2.6 – Диаграмма детализации основного процесса в нотации IDEF0

Детализируем процесс создания упражнений. Для этого выделим варианты взаимодействий пользователя с системой при создании упражнения. Результат выделения представлен в таблице 2.2.

Таблица 2.2 – Детализация процесса создания упражнений

Название процесса	Входные данные	Управляющие данные	Механизм	Результат процесса
Формирование списка упражнений	Выбранные группы словарей, словари, переводы	Запрос на создание упражнения	Система формирования упражнений	Список упражнений
Получение переводов упражнений	Список упражнений	Запрос на создание упражнения	Система формирования упражнений	Упражнения и переводы для них
Формирование правильных и неправильных ответов	Упражнения и переводы	Получение списка переводов упражнения	Система формирования упражнений	Подборка упражнений с правильными и неправильными ответами

На основе полученных процессов составим диаграмму детализации процесса формирования упражнений в нотации IDEF0. Результат изображен на рисунке 2.7.

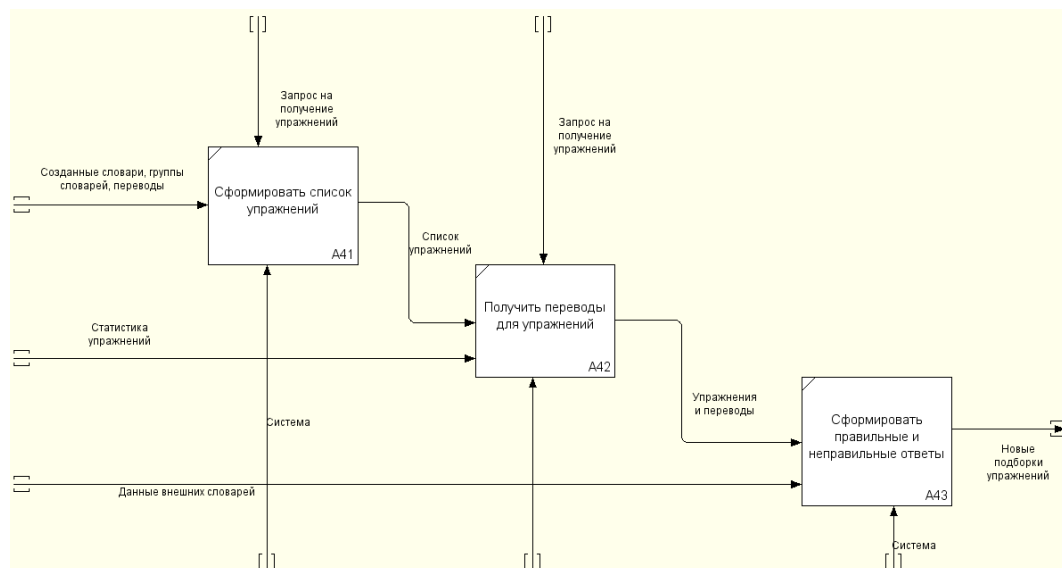


Рисунок 2.7 – Диаграмма детализации процесса формирования упражнений в нотации IDEF0

## 2.3 Разработка алгоритма функционирования приложения

Для создания алгоритма функционирования приложения выделим список действий приложения и порядок их следования друг за другом. Затем изобразим результаты анализа в графически в виде диаграмм нотации IDEF3. Выделенные действия представлены в таблице 2.3. Порядок следования действий друг за другом приведен в таблицах 2.4 и 2.5.

Таблица 2.3 – Список действий и объектов, составляющих процесс

№ действия	Название действия
1	Работа со словарем
2	Регистрация
3	Создание группы словарей
4	Создание словаря
5	Получение упражнений
6	Создание группы словарей
7	Формирование списка предложений
8	Получение переводов для упражнений
9	Формирование правильных и неправильных ответов
10	Получение упражнений

Таблица 2.4 – Список действий с указанием предшествующих и последующих событий с указанием типа связи

№ или номера предшествующих действий	Тип связи	№ действия	Тип связи	№ или номера последующих действий
		1		
		2	Временное предшествование	3,4
3,4	Объектный поток	5		
		6	Объектный поток	7
7	Объектный поток	8,9	Объектный поток	10
		10		

Таблица 2.5 – Список действий с указанием предшествующих и последующих событий с указанием установленных отношений

№ или номера предшествующих действий	Тип связи	№ действия	Тип связи	№ или номера последующих действий
2	Асинхронный &	3,4	Асинхронный &	5
7	Асинхронный &	8,9	Синхронный &	10

Изобразим полученный в результате составления порядка событий при взаимодействии пользователя с системой алгоритм в виде нотации IDEF3. Полученные диаграммы изображены на рисунках 2.8 – 2.10.

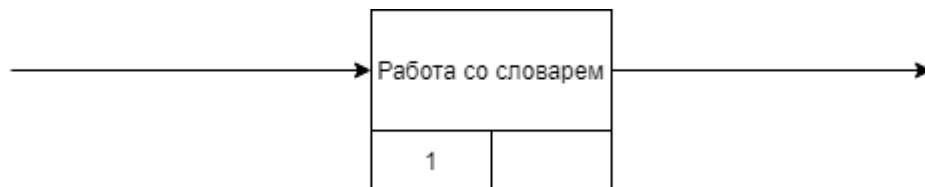


Рисунок 2.8 – Диаграмма первого уровня взаимодействия пользователя с системой в нотации IDEF3



Рисунок 2.9 – Диаграмма декомпозиции первого уровня взаимодействия пользователя с системой в нотации IDEF3



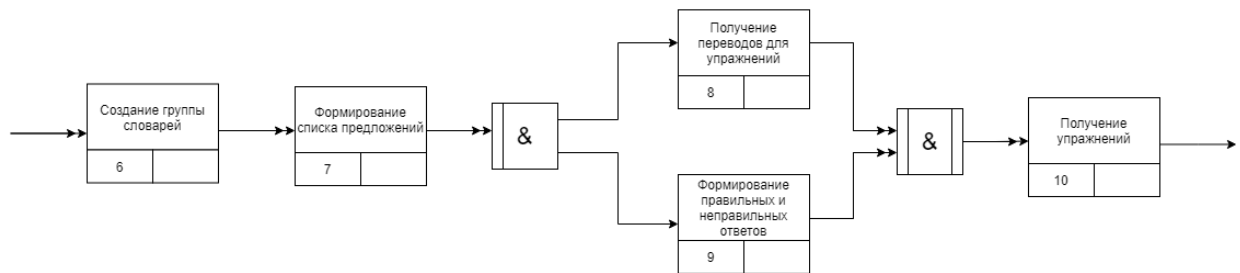


Рисунок 2.10 – Диаграмма декомпозиции действия № 5 «Получение упражнений» в нотации IDEF3

Расширим модель представления алгоритма с точки зрения бизнес-процессов. Используем для представления результатов анализа данной области графическую нотацию BPMN. Покажем какой набор действий системы необходимо выполнить для решения каждой бизнес-задачи, решаемой системой, а также участников этих действий и объекты данных, используемые в них. Результат анализа представлен в таблице 2.6.

Таблица 2.6 – Список задач, действующих лиц и объектов данных

№ задачи	Название задачи	№ и список действий, составляющих решение задачи	Участник, осуществляющий решение задачи	Объекты данных
1	Регистрация пользователя	Заполнение и отправка формы регистрации	Пользователь	БД Пользователей
2	Просмотр словарей	Выбор группы словарей, выбор словаря	Пользователь	БД словарей
3	Редактирование словарей	Выбор группы словарей, выбор словаря, выбор перевода, редактирование	Пользователь	БД словарей
4	Формирование упражнений	Выбор группы словарей или словаря	Пользователь	БД словарей

Используя результаты анализа бизнес-процессов, изобразим диаграммы бизнес-процессов в нотации IDEF3. Результаты изображены на рисунках 2.11 и 2.12.

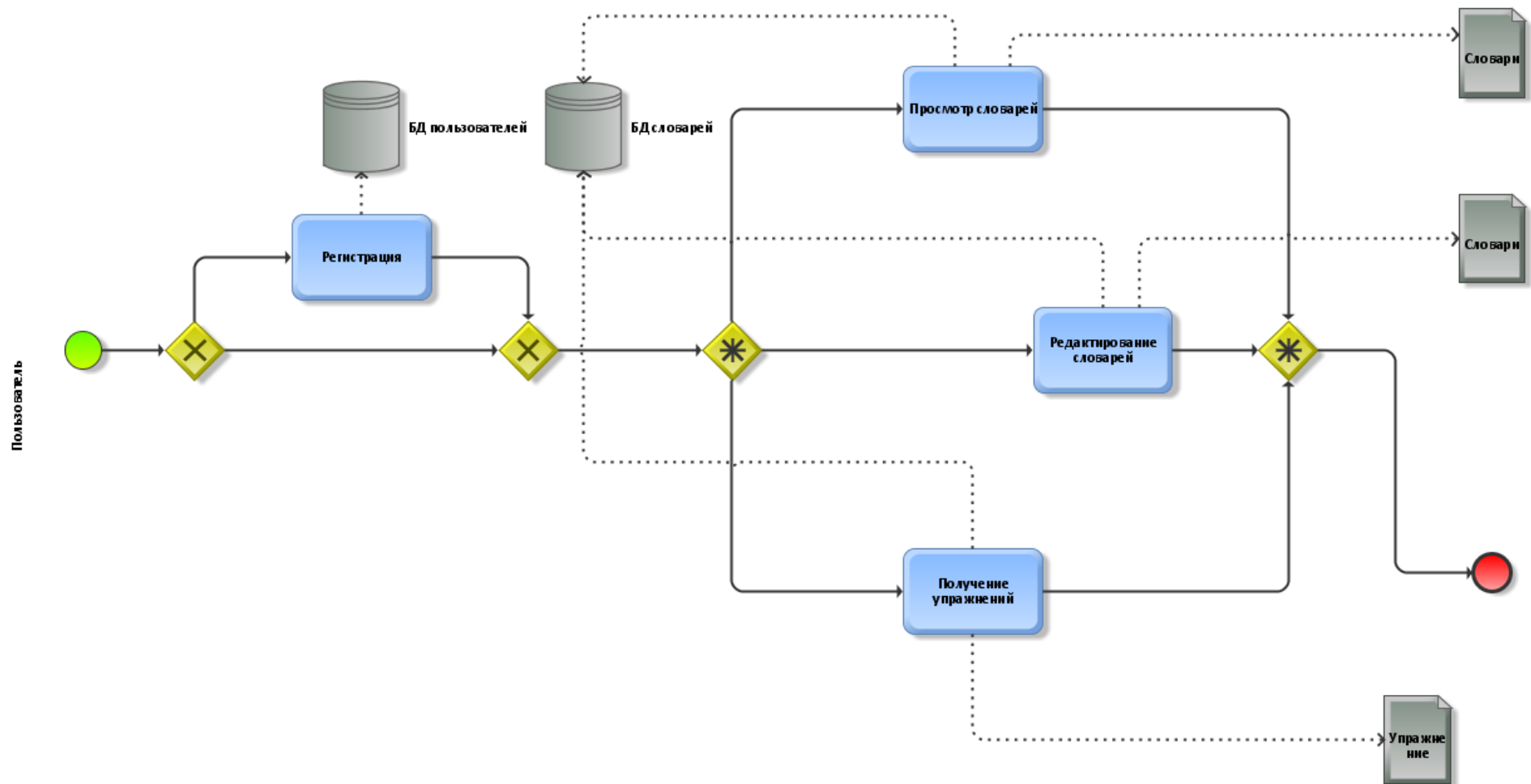


Рисунок 2.11 – Упрощенная диаграмма бизнес-процессов в нотации BPMN

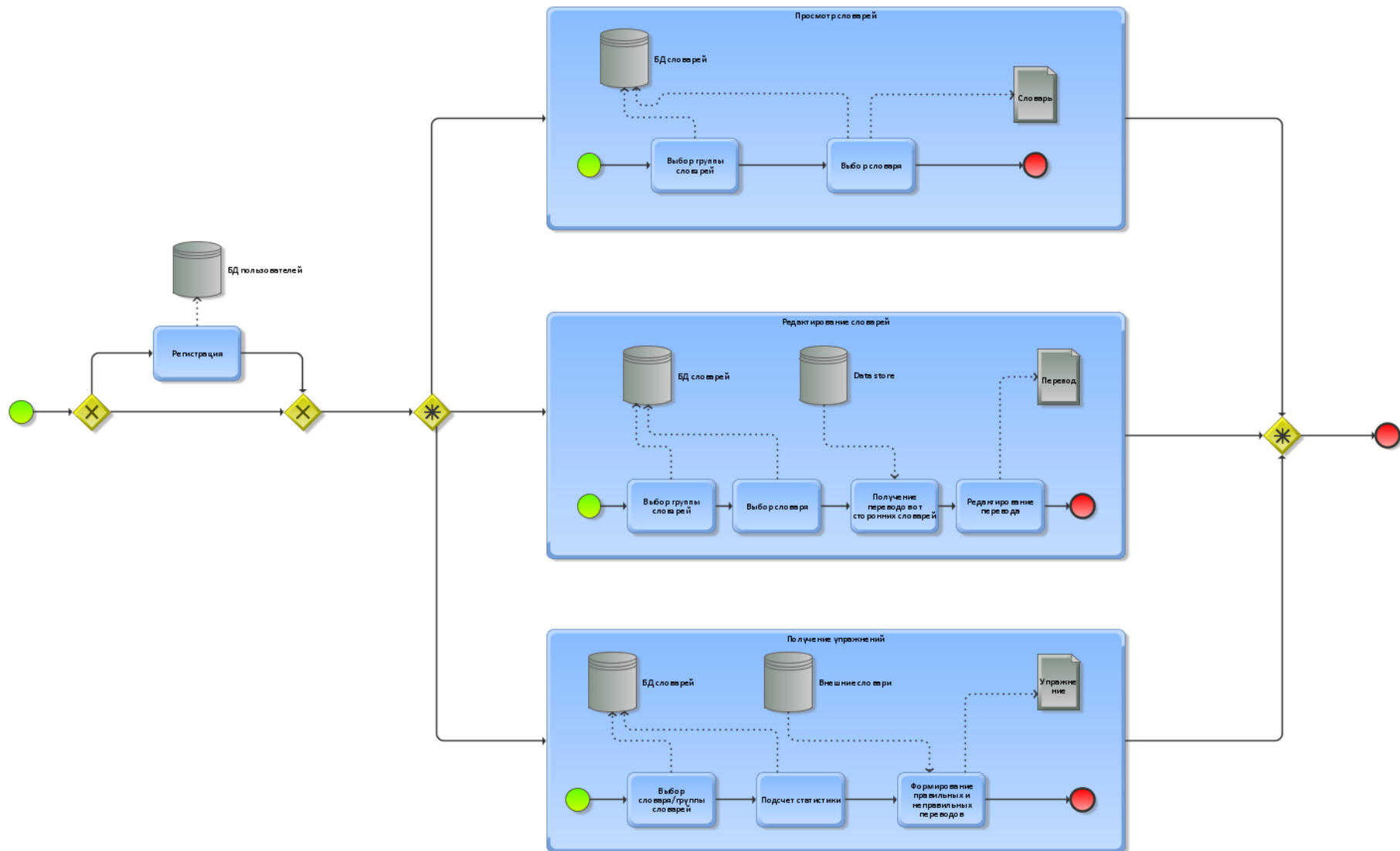


Рисунок 2.12 – Детализированная диаграмма бизнес-процессов в нотации BPMN

## 2.4 Разработка структуры данных

Для описания структуры базы данных выделим сущности, присутствующие в системе:

1. **Группа словарей.** Содержит информацию о группе словарей. Словари делятся на группы для классификации.

2. **Словарь.** Содержит информацию о словаре. Используется для хранения переводов.

3. **Перевод.** Содержит перевод: переводимое слово (выражение, предложение), его выбранные переводы, флэш-карточку (контекстную информацию).

4. **Единица перевода.** Содержит одну единицу перевода (на родном языке) для того, чтобы переводы могли содержать список таких единиц.

5. **Выбранный перевод.** Содержит связь между сущностями «Единица перевода» и «Перевод».

6. **Пользователь.** Содержит информацию о пользователе системы.

7. **Тип упражнения.** Содержит все возможные типы упражнений.

8. **Упражнение.** Содержит информацию о том, какое упражнение пользователь выполняет (выполнил), список вариантов ответа.

9. **Ответ пользователя при упражнении.** Содержит информацию об ответе пользователя при прохождении упражнения по переводу.

10. **Экспорт группы словарей.** Содержит информацию о том, какому пользователю был предоставлен доступ к каким словарям или группам словарей.

Для этих сущностей выделим атрибуты:

1. **Группа словарей.** идентификатор, дата удаления, идентификатор пользователя, название, описание.

2. **Словарь.** идентификатор, дата удаления, идентификатор группы словарей, название, описание.

3. **Перевод.** идентификатор, дата удаления, идентификатор словаря, переводимое выражение, пользовательский контекст использования.

4. **Единица перевода.** идентификатор, название единицы перевода.

5. **Выбранный перевод.** идентификатор перевода, идентификатор единицы перевода.

6. **Пользователь.** идентификатор, дата удаления, логин, пароль, электронная почта.

7. **Тип упражнения.** идентификатор, название типа упражнения.

8. **Упражнение.** идентификатор, идентификатор типа упражнения, идентификатор перевода, идентификатор пользователя.

9. **Ответ на упражнение.** идентификатор, идентификатор упражнения, идентификатор единицы перевода, признак правильности выбора.

10. **Экспорт группы словарей.** идентификатор пользователя, идентификатор группы словарей.

Опишем связи между сущностями на естественном языке:

1. Каждая **группа словарей** может содержать несколько **словарей**.
2. Каждый **словарь** может содержать несколько **переводов**.
3. Каждый **перевод** может содержать несколько **выбранных переводов**.
4. Каждый **выбранный перевод** относится только к одной единице перевода.
5. Каждый **пользователь** имеет несколько **групп словарей**.
6. Каждое **упражнение** относится только к одному **типу упражнений**.
7. Каждый **ответ на упражнение** является ответом только на одно **упражнение**.
8. Каждый **экспорт группы словарей** относится только к одному **пользователю**.
9. Каждый **экспорт группы словарей** относится только к одной **группе словарей**.

Составим матрицу отношений между выделенными сущностями на основе естественного описания связей между сущностями. Матрица отношений представлена в таблице 2.7.

Приведем использованные модели базы данных к 3 нормальной форме. Используя полученные отношения между сущностями и атрибуты сущностей составим диаграмму структуры базы данных в нотации IDEF1X. Результат изображен на рисунке 2.13.

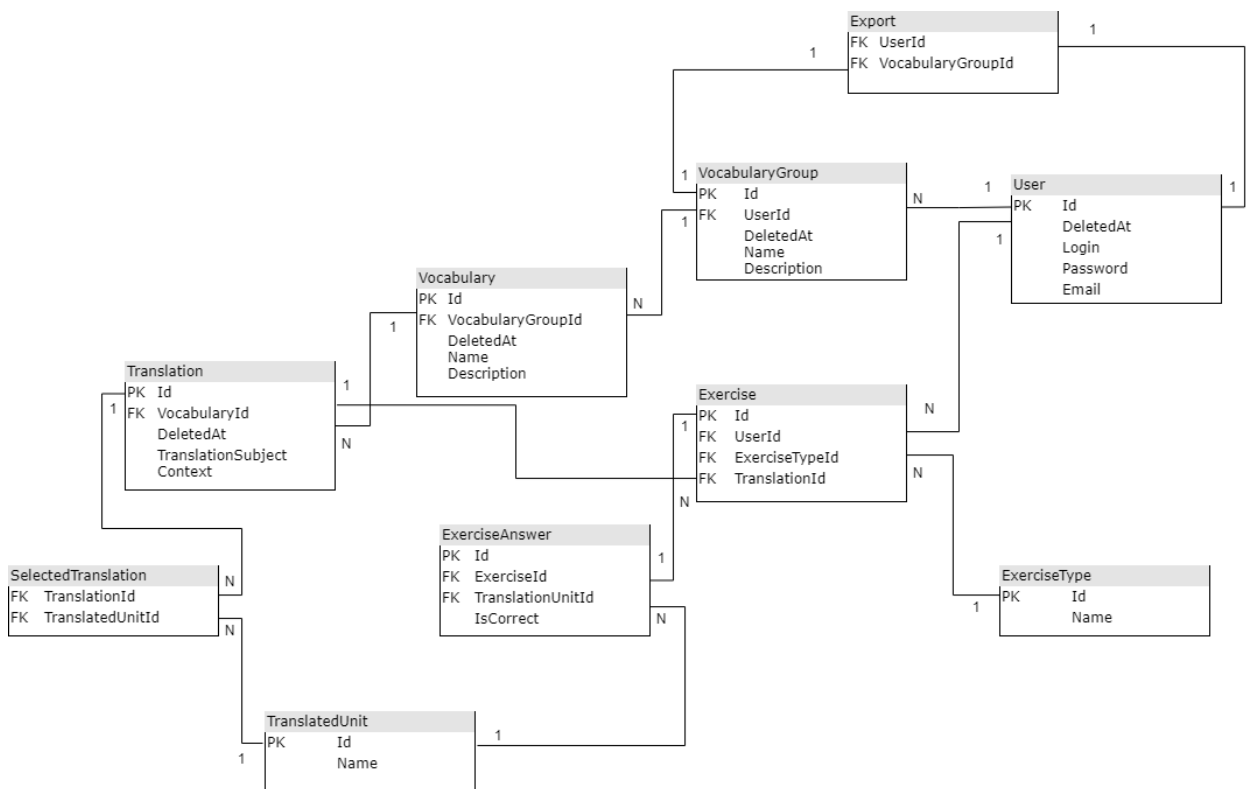


Рисунок 2.13 – Структура базы данных разрабатываемого приложения в графической нотации IDEF1X

Таблица 2.7 – Матрица отношений между сущностями проектируемой системы

–	Группа словарей	Словарь	Перевод	Выбранный перевод	Единица перевода	Пользователь	Тип упражнения	Упражнение	Ответ на упражнение	Экспорт
Группа словарей	–	содержит (1:N)				содержится (1:1)				относится (1:1)
Словарь	относится (1:1)	–	содержит (1:N)							
Перевод		относится (1:1)	–	содержит (1:N)				содержит (1:N)		
Выбранный перевод			относится (1:1)	–	относится (1:1)					
Единица перевода			относится (1:1)	относится (1:1)	–				относится (1:1)	
Пользователь	содержит (1:N)					–		содержит (1:N)		относится (1:1)
Тип упражнения							–	содержит (1:N)		
Упражнение			относится (1:1)				является (1:1)	–	относится (1:1)	
Ответ на упражнение					относится (1:1)			относится (1:1)	–	
Экспорт	относится (1:1)					относится (1:1)				–

## **Выводы к разделу 2**

В данном разделе была спроектирована архитектура и внутреннее устройство разрабатываемого приложения. Были описаны внешние сущности приложения, сценарии их взаимодействия с разрабатываемой системой; данные, которые передаются в процессе различных действий пользователя между структурными компонентами системы; последовательности событий, случающихся в системе при совершении пользователем всех возможных действий; основные сущности, составляющие данные приложения, их атрибуты и отношения между ними.

На каждом этапе были сформированы наглядные графические представления результатов проектирования. Для представления потоков данных в приложении использовалась нотация DFD; для описания процессов взаимодействия пользователя с приложением – нотация IDEF0, для представления алгоритма процессов приложения – нотация IDEF3; для описания бизнес-процессов нотация BPMN; для описания структуры данных приложения – нотация IDEF1X.

После проведения проектирования в данном разделе можно приступать к разработке приложения.



### **3 РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ «АДАПТИВНАЯ СИСТЕМА В ПОМОЩЬ ИЗУЧАЮЩЕМУ ИНОСТРАННЫЙ (АНГЛИЙСКИЙ) ЯЗЫК» И ЕЕ ПОДСИСТЕМ**

#### **3.1 Выбор и обоснование информационного обеспечения ИС**

Для автоматизации перевода лексики, формирования примеров использования лексики в контексте иностранного языка, формирования определений иностранной лексики необходимо использовать внешние источники. Далее рассмотрим некоторые популярные API электронных переводчиков и словарей, которые можно использовать для получения из них данных разрабатываемым приложением.

##### **3.1.1 Обзор переводчика Google Translate API**

Google Translate API позволяет быстро перевести произвольную строку текста с одного языка на другой. Он позволяет использовать мощную облачную технологию машинного обучения Google для автоматического перевода текстов с одного языка на другой. Кроме того, если исходный язык неизвестен, API может определить предоставленный язык. Технология, лежащая в основе API, постоянно обновляется с учетом улучшений исследовательских групп Google, что приводит к усовершенствованию услуг перевода. Google Translate API поддерживает более 100 языков. Сервис доступен бесплатно с ограничением в 50 запросов в день.

Как и другие продукты Google, Translate API хорошо документирован с множеством примеров интеграции, что упрощает его использование. Кроме того, существует активное сообщество, которое поможет вам с любыми проблемами интеграции. К сожалению, сервис не доступен в Крыму, так что возможности по его использованию сильно ограничены.

Google Translate API доступен по адресу [4] и состоит из двух эндпоинтов (endpoint) HTTP POST:

- `translate`: принимает переводимую строку, язык, на который нужно перевести эту строку и, опционально, язык исходной строки и формат исходной строки (HTML или простой текст); возвращает JSON-объект со свойством “`translations`”, который содержит список переводов;

- `detect`: принимает только строку, для которой нужно определить язык, в котором она написана; возвращает JSON-объект со свойством “`detections`”, который содержит список выявленных языков в порядке вероятности достоверности определения.

### 3.1.2 Обзор переводчика Microsoft Text Translation API

Microsoft Text Translation API доступен в рамках набора сервисов Microsoft Azure и позволяет использовать технологию машинного перевода Microsoft для создания приложений, способных обеспечивать многоязычную поддержку. Как и Google Translate API, сервис от Microsoft позволяет кроме непосредственно перевода определить язык исходного текста с помощью технологий машинного обучения. Кроме того, он способен производить транслитерацию слов и предложений из одного сценария в другой, а также возможности двуязычного словаря для отображения альтернативных переводов. с или на английский язык. Microsoft Text Translation 3.0 API поддерживает более 90 языков. Сервис доступен бесплатно при менее 2500 запросах в месяц, при превышении этого лимита следует использовать один из платных планов подписки.

Microsoft предоставляет краткие руководства, учебные пособия и другие полезные ресурсы, которые помогут вам максимально эффективно использовать API.

Microsoft Text Translation 3.0 API доступен по адресу [5] и состоит из следующих HTTP POST эндпоинтов:

- `translate`: принимает строку, которую необходимо перевести, языки, на который производится перевод и, опционально, язык исходной строки, ее

формат (HTML или простой текст); возвращает JSON-объект с определенным языком и массивом списка переводов;

- detect: принимает строку с текстом, язык которого необходимо определить; возвращает JSON-массив переводов с вероятностями правильности определения;

- transliterate: позволяет переписать исходный текст с помощью букв из другого алфавита, но доступен не для всех комбинаций алфавитов; интереса в рамках данной работы не представляет и не рассматривается подробно;

- breakSentence: позволяет выделить из исходного текста границы предложений; интереса не представляет и не рассматривается подробно;

- dictionaryLookup: позволяет получить список альтернативных переводов заданного слова на заданный язык; возвращает массив в формате JSON, содержащий список переводов с соответствующими им вероятностями корректности и количествами доступных примеров использования и идиом;

- dictionaryExamples: позволяет для заданного слова и одной из альтернатив его перевода на определенный язык получить список примеров его использований в контексте на этом языке; принимает строку исходного слова, строку его перевода и язык, примеры из которого необходимо получить; возвращает JSON-массив, содержащий список примеров и идиом с данным словом; используется вместе с dictionaryLookup и представляет повышенный интерес в рамках данной работы.

### 3.1.3 Обзор переводчика MyMemory

MyMemory – это обширная онлайн-база данных, содержащая миллиарды слов, переведенных профессиональными переводчиками. С помощью MyMemory Translation API вы можете искать переводы, доступные в обширной базе данных MyMemory. Переводы ранжируются по качеству и сходству, что позволяет вам получить лучший перевод для данного исходного текста. Если человеческий перевод недоступен, API предоставляет современный машинный перевод. Кроме того, вы можете использовать API

для вставки переводов в базу данных MyMemory. Особенностью данного переводчика является специализация на переводе отдельных слов и словосочетаний.

API сервиса MyMemory доступен по адресу [6] и состоит из следующих эндпоинтов:

- get – производит поиск переводов выбранного слова по базе MyMemory; принимает исходную строку, язык исходной строки, язык, на который необходимо произвести перевод и флаг, который указывает, использовать ли машинный перевод; возвращает JSON-массив со списком переводов исходной строки;

- set – позволяет загрузить в базу MyMemory собственный перевод; принимает исходный текст, его перевод, исходный язык и язык, на который производится перевод.

### **3.1.4 Обзор переводчика Amazon Translate**

Amazon Translate – это сервис нейронного машинного перевода, обеспечивающий быстрый, высококачественный, перевод с пользовательскими настройками с одного языка на другой. Amazon Translate позволяет настраивать результаты машинного перевода за счет возможностей Custom Terminology и Active Custom Translation. С помощью Custom Terminology можно определить, как переводятся названия торговых марок, названия моделей и другие уникальные термины. С помощью Active Custom Translation можно сгенерировать настроенный машинный перевод, адаптированный к потребностям вашей отрасли. Кроме широкой документации к самому API, Custom Terminology и Active Custom Translation, Amazon предоставляет широкую индивидуальную поддержку для пользователей сервиса. Бесплатно доступны 2 миллиона символов в месяц в течение 12 месяцев. При пересечении бесплатной границы по шкалам времени или количества символов, сервис становится доступен за 15 долларов США за каждый миллион символов.

Amazon Translate доступен только при использовании Amazon Web Services. Рассмотрим доступные эндпоинты сервиса:

– `translateText`: производит перевод заданного текста в заданный язык; принимает строку исходного текста, язык исходного текста, язык, в который текст необходимо перевести; возвращает JSON-объект со свойством `TranslatedText`, которое содержит перевод.

По описанию спецификации API перевода Amazon Translate можно сделать вывод, что сервис специализируется на переводе больших текстов, электронных писем и документов. Следовательно, для перевода отдельных слов его может быть избыточно использовать.

### **3.1.5 Обзор переводчика Translate.Yandex**

Yandex Translate API – отечественный сервис для перевода. Он поддерживает 90 языков и умеет переводить отдельные слова и целые тексты. С помощью API можно получить доступ к сервису машинного перевода, выполненному с помощью технологий машинного обучения. Сервис доступен как часть группы сервисов «Облако Яндекс» по цене 50 рублей за каждые 100.000 символов.

Yandex Translate API доступен по адресу [7] и состоит из следующих HTTP POST эндпоинтов:

– `translate` – используется для перевода текста на указанный язык; принимает в теле запроса следующие параметры: список текстов на перевод, необязательный язык исходных текстов, язык результирующих текстов; возвращает JSON-массив, содержащий список переводов с соответствующими определенными языками.

Далее рассмотрим возможные сервисы, которые можно использовать для получения разъяснений значений лексики и получения примеров этой лексики в контексте языка.

### 3.1.6 Обзор электронного словаря Oxford Dictionaries

Oxford Dictionaries API дает доступ к крупнейшему словарю постоянно растущего списка языков. Сервис использует интенсивную программу языковых исследований собственной разработки для поддержания актуальности, точности и надежности данных. Oxford Dictionaries поддерживает 35 языков. Сервис позволяет получить определения, произношения, происхождения, грамматические особенности слов, курированные коллекции использования этих слов в контексте языка, статистику использования этих слов. Oxford Dictionaries доступен за 0,002 фунтов за запрос.

Oxford Dictionaries API доступен по адресу [8] и содержит следующие HTTP GET эндпоинты:

- words – производит поиск определений и примеров использования слова в контексте языка; принимает слово и язык; возвращает JSON-объект с массивами определений и примеров использования слова;
- sentences – производит поиск предложений живого языка с использованием данного слова, включая книги, журналы, газеты и контент из интернета; принимает идентификатор слова, который может быть получен с помощью других эндпоинтов предоставляемого API; возвращает JSON-массив предложений, включающих данное слово;
- thesaurus – позволяет получить похожие и противоположные по значению слова к запрашиваемому; принимает идентификатор слова, который может быть получен с помощью других эндпоинтов предоставляемого API; возвращает JSON-массив
- translations – позволяет получить переводы заданного слова; принимает исходный язык слова, идентификатор слова, язык, на который слово необходимо перевести; возвращает JSON-массив, содержащий список переводов заданного слова, а если переводов в базе не присутствует – определение этого слова.

### 3.1.7 Обзор словаря Macmillan Dictionary

Macmillan Dictionary API предоставляет доступ ко многим возможностям одноименного известного словаря. Для английской и американской версии доступны следующие данные: определения, произношение, грамматические особенности лексики, примеры предложений с указываемой лексикой в контексте языка. Сервис доступен бесплатно для некоммерческого использования.

Macmillan Dictionary API доступен по адресу [9] и содержит следующие эндпоинты:

- dictionaries (или dictionaries/search) – позволяет получить данные определенного словаря; принимает строку для поиска по словарю; возвращает JSON-объект с данными определенного словаря, в том числе его идентификатор для получения лексики, содержащейся в нем через другие эндпоинты предоставляемого API.

- entries – позволяет получить всю информацию, связанную с лексической единицей, содержащейся в словаре; принимает идентификатор словаря, идентификатор лексической единицы, которые могут быть получены с помощью других эндпоинтов Macmillan Dictionary API; возвращает JSON-объект, содержащий массивы с определениями, произношением, грамматикой, примерами предложений лексики.

### 3.1.8 Выбор словаря и переводчика

Выберем сервисы, которые будем использовать в ходе разработки. Сторонними сервисами необходимо покрыть три случая: получение списка переводов вводимой лексической единицы, получение списка определений лексической единицы и получение ее примеров использования в контексте иностранного языка. Сервисы Oxford Dictionaries и Macmillan Dictionary API позволяют удовлетворить два последних случая. Из них выбор сделаем в пользу Macmillan Dictionary т.к. он позволяет получить все необходимые данные для данных обоих случаев за меньшее количество запросов и является

бесплатным. Для удовлетворения первого случая невозможно использовать Google Translate API т.к. он заблокирован в Республике Крым. Кроме того, сервис Amazon Translate является для данного варианта использования избыточным. Переводчик MyMemory исключим из списка т.к. он слабо подходит для перевода текста длиной более 3 слов, что может в нашем приложении понадобиться. Между Yandex Translate, и Microsoft выбор сделаем в пользу отечественного сервиса Yandex Translate т.к. он более доступен и прост в использовании.

### **3.2 Выбор и обоснование языка программирования и инструментальных средств для разработки**

При текущем уровне развития отрасли разработки ПО при выборе инструментальных средств для создания программных продуктов следует принимать во внимание не только достоинства и недостатки языков программирования, но и степень развитости инфраструктуры, с помощью которой производится разработка на этом языке, библиотек в открытом доступе, фреймворков для реализации разного рода задач и функциональность и удобство доступных сред разработки.

Для реализации приложения была выбрана платформа .NET Core, разработка на которой ведется преимущественно с помощью языков программирования C# и F#. На платформе основным фреймворком для создания WEB-приложений является ASP.NET Core, фреймворком для доступа к базе данных – Entity Framework Core (далее EF Core). Главной СУБД в экосистеме .NET является Microsoft SQL Server, с которой легко интегрируется фреймворк EF Core. Для разработки используем интегрированную среду разработки Rider от компании JetBrains так как она имеет встроенные функции расширения ReSharper (расширение для стандартной среды разработки на .NET Visual Studio, которое привносит в нее



функциональность сред разработки от JetBrains: IDEA и другие). Рассмотрим платформу .NET Core, язык C#, который будет использоваться для создания приложения, фреймворки ASP.NET Core и EF Core и СУБД SQL Server.

### **3.2.1 Обоснование использования платформы .NET Core**

**.NET Core** – модульная платформа для разработки программного обеспечения с открытым исходным кодом. Появилась в 2016 году как развитие платформы .NET. .NET Core является, по сути, полностью переписанной .NET платформой с изменениями на всех уровнях. Целью было позволить .NET приложениям выполняться не только на операционной системе Windows, но и на других операционных системах. Кроме этого, репозиторий с программным кодом новой платформы изначально был доступен как Open Source проект, что значительно увеличивает надежность платформы. Основой обеих платформ является Common Language Runtime (CLR), которая является средством, способным объединить в рамках одного проекта модули, написанные на нескольких языках программирования. Поэтому модули, написанные на двух главных языках платформ – C# и F# - могут быть свободно вызваны друг из друга. Из этого вытекает богатство доступных на этих языках библиотек. На момент написания работы актуальной версией является .NET 5 (суффикс Core упущен производителем, компанией Microsoft, намеренно как констатация развития всего .NET мира в Core направлении), с момента выхода платформы вышло 4 номерных версии, 3 из которых будут поддерживаться длительно, и новые версии продолжают появляться каждый год. В целом, платформа за последние годы значительно развилась и продолжает развиваться, а количество разработчиков, использующих эту платформу, постоянно растет, поэтому .NET Core является одной из самых актуальных.

### **3.2.2 Обоснование использования языка программирования C#**

Язык программирования C# был разработан в далеком 2001 году в компании Microsoft под руководством Мартина Хейлсберга, участвовавшего до

этого в разработке языка Delphi, как язык исключительно для платформы .NET. С тех пор было выпущено 9 мажорных версий. C# (произносится как «си шарп») — современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать множество видов безопасных и надежных приложений, работающих в экосистеме .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. C# — это объектно- и компонентно-ориентированный язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО. Вот лишь несколько функций языка C#, которые позволяют создавать надежные и устойчивые приложения. Сборка мусора автоматически освобождает память, занятую недоступными неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. Лямбда-выражения поддерживают приемы функционального программирования. Синтаксис LINQ создает общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем. Учитывая активное развитие платформы в целом и языка в частности, язык C# является отличным выбором для создания как WEB-приложений, так и других видов приложений для настольных и мобильных операционных систем.

### **3.2.3 Обоснование использования фреймворка ASP.NET Core**

**ASP.NET Core** является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания

современных облачных приложений, подключенных к Интернету. ASP.NET Core позволяет выполнять следующие задачи:

- создавать веб-приложения и службы, приложения Интернета вещей (IoT) и серверные части для мобильных приложений;
- использовать избранные средства разработки в Windows, macOS и Linux;
- выполнять развертывания в облаке или локальной среде.

Миллионы разработчиков использовали и продолжают использовать ASP.NET 4.x для создания веб-приложений. ASP.NET Core — это модификация ASP.NET 4.x с архитектурными изменениями, формирующими более рациональную и более модульную платформу. ASP.NET Core предоставляет следующие преимущества:

- единое решение для создания пользовательского веб-интерфейса и веб-API;
- разработано для тестируемости;
- Razor Pages упрощает написание кода для сценариев страниц и повышает его эффективность;
- Blazor позволяет использовать в браузере язык C# вместе с JavaScript. совместное использование серверной и клиентской логики приложений, написанных с помощью .NET;
- возможность разработки и запуска в ОС Windows, macOS и Linux;
- открытый исходный код и ориентация на сообщество;
- интеграция современных клиентских платформ и рабочих процессов разработки;
- поддержка размещения служб удаленного вызова процедур (RPC) с помощью gRPC;
- облачная система конфигурации на основе среды;
- встроенное введение зависимостей;
- упрощенный высокопроизводительный модульный конвейер HTTP-запросов.

### 3.2.4 Обоснование использования фреймворка Entity Framework Core

**Entity Framework (EF) Core** - представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами. EF Core может использоваться в качестве объектно-реляционного модуля сопоставления (O/RM), который:

- позволяет разработчикам .NET работать с базой данных с помощью объектов .NET;
- устраняет необходимость в большей части кода для доступа к данным, который обычно приходится писать;
- EF Core поддерживает множество систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер. В данный момент существует 22 провайдера, в том числе провайдеры для самых популярных СУБД: SQL Server, MySQL, PostgreSQL, Sqlite, Oracle и др.

### 3.2.5 Обоснование использования СУБД Microsoft SQL Server

**Microsoft SQL Server** – система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется

для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка. SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке. SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux. SQL Server характеризуется такими особенностями как:

- производительность: SQL Server работает очень быстро;
- надежность и безопасность: SQL Server предоставляет шифрование данных;
- простота: с данной СУБД относительно легко работать и вести администрирование.

### **3.2.6 Обоснование использования библиотеки ReactJS**

Для создания клиентского приложения используем библиотеку ReactJS, разработанную компанией Facebook. React – гибкая JavaScript библиотека с открытым исходным кодом для создания клиентских приложений и пользовательских интерфейсов в браузере. Она позволяет создавать собирать сложный UI из маленьких изолированных кусочков кода, называемых «компонентами». Особенностью библиотеки является использование реактивного подхода, который и позволяет реактивно декларативно описывать программные модули. Библиотека является самой популярной в мире клиентской разработки, из-за чего поиски уже реализованных компонентов не составляет проблем.

### 3.3 Общее описание архитектуры системы

Большинство современных WEB-приложений интерактивны, что требует возможности создания богатой логики на клиентской стороне. Самой удобной архитектурой для обеспечения такой возможности является SPA (Single Page Application). Архитектура заключается в том, что при первом посещении приложения пользователем он загружает все клиентское приложение сразу, а данные, которые нужны для работы приложения, загружаются только по мере надобности с сервера. Таким образом, большая часть логики переносится с сервера на клиент, что сильно усложняет приложение. На рисунке 3.1 изображена схема жизненного цикла SPA приложения.

Доминирующим языком программирования на стороне клиента является JavaScript и большинство клиентских приложений написаны именно на нем. Главным способом передачи данных с сервера на клиентскую сторону при такой архитектуре является HTTP запрос, отправляемый с клиентской стороны с помощью технологий XMLHttpRequest, JQuery Ajax и Fetch API. Данные при этом передаются в формате JSON, который позволяет представлять сложные структуры данных, такие как объекты и массивы с неограниченной вложенностью, в виде строки.

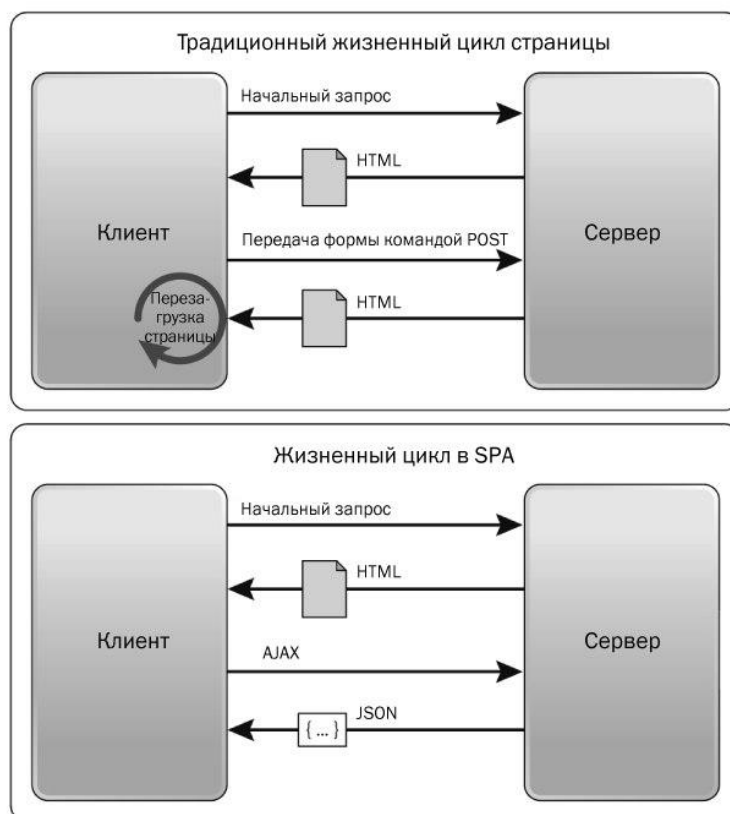


Рисунок 3.1 – Схема жизненного цикла SPA приложения

Сервер при данной архитектуре берет на себя роль авторизации, аутентификации и доступа к данным.

### 3.4 Программная реализация информационной системы

#### 3.4.1 Создание решения в Rider

Для создания решения необходимо совершить следующие действия: перейти в подменю «Файл», затем выбрать тип проекта «ASP.NET Core Web Application», затем указать 5.0 версию .NET, нажать кнопку «Create». На рисунке 3.1 показано контекстное окно создания решения в JetBrains Rider. После создания получена следующая структура приложения, которой мы будем придерживаться в разработке:

1. Файлы Program.cs и Startup.cs. Все приложения .NET Core по соглашению должны иметь точку входа в виде метода Main класса Program. В этом месте в приложении ASP.NET Core создается Host - абстракция для инкапсуляции всех ресурсов приложения: реализация HTTP сервера, конфигурация сервера, компоненты конвейера, сервисы инверсии зависимостей (DI), логирование. Startup.cs - класс, в котором настраиваются сервисы, используемые приложением (метод ConfigureServices), настраивается конвейер обработки HTTP запросов как список промежуточных компонентов middleware (метод Configure).

2. Папка Controllers. Папка содержит контроллеры – компоненты архитектуры MVC или сущности, предоставляющие HTTP эндпоинты приложения.

3. Папка Data. Папка содержит все, что в приложении связано с созданием модели данных EF Core. Это классы описания моделей (builder) и классы контекста базы данных – сущности для инкапсуляции в них всей логики работы с базой данных с помощью EF Core.

4. Папка Models. Папка содержит модели данных, которым сопоставляются сущности базы данных.

5. Папка ViewModels. Папка содержит вью-модели – объекты, которые представляют данные, которые сервер передает на клиентскую часть.



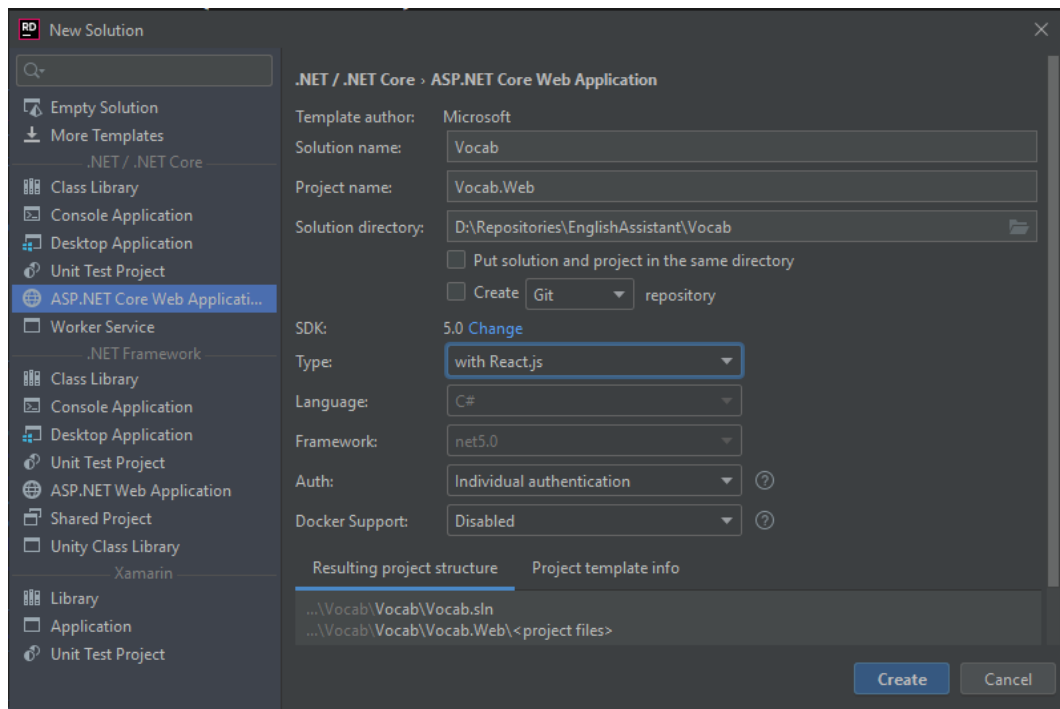


Рисунок 3.2 – Контекстное окно создания решения в JetBrains Rider

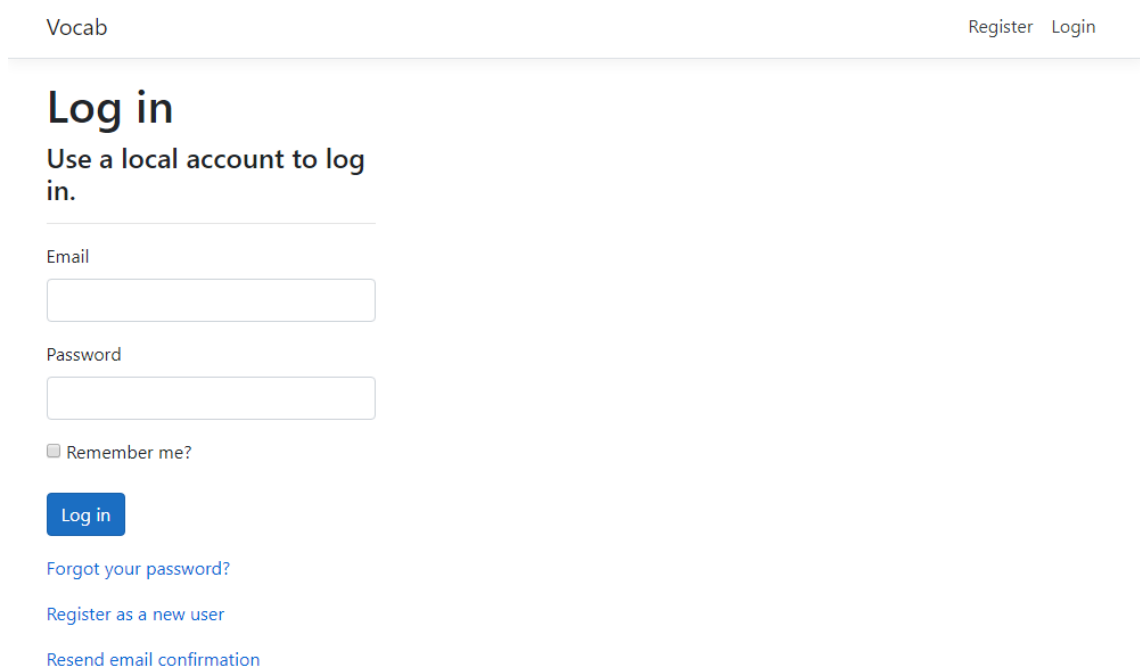
### 3.4.2 Реализация системы аутентификации и авторизации

Система аутентификации и авторизации должна быть выполнена на серверной стороне приложения. Для этого используем библиотеку Identity, которая предоставляет все нужные функции и легко интегрируется в ASP.NET Core. Для доступности нужных функций добавим пакет с библиотекой “Microsoft.AspNetCore.Identity.EntityFrameworkCore” и ее зависимостями в список зависимостей разрабатываемого приложения. Это делается с помощью следующей команды командной строки: “dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore”. После этого необходимо создать пользовательскую модель и связать ее с базой данных с помощью Entity Framework. Для этого создадим файл и класс “AppIdentityDbContext”, который должен быть унаследован от `ApiAuthorizationDbContext<User>` (так как в приложении используется OAuth с использованием токенов аутентификации). Далее необходимо подключить созданные классы контекста и сервисы аутентификации в список сервисов приложения (находится в классе

Startup.cs) вызовом соответствующих методов: `AddDefaultIdentity`, `AddAuthentication` и `AddIdentityServerJwt`.

После добавления нужных сервисов в приложение произведем настройку Identity создав экземпляр класса `IdentityOptions` и задав его как параметр сервиса аутентификации. Зададим для приложения возможность регистрации новых пользователей, максимальное количество безуспешных попыток авторизации, требования к паролю и другие.

После этого нужно только создать представления, соответствующие всем действиям пользователей при работе с приложением. На рисунках 3.3 – 3.5 представлены полученные страницы.



Vocab Register Login

## Log in

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Рисунок 3.3 – Форма логина приложения

Система работает при помощи реализованной в Identity системы генерации специального токена в формате JWT (JSON Web Token), который однозначно идентифицирует пользователя и используется для аутентификации всех его запросов на сервер.

Vocab Hello astro6703@gmail.com! Logout

---

## Manage your account

### Change your account settings

[Profile](#)  
[Email](#)  
[Password](#)  
[Two-factor authentication](#)  
[Personal data](#)

#### Profile

Thank you for confirming your email. ×

Username

Phone number

Save

Рисунок 3.4 – Страница личного кабинета приложения

Vocab Hello astro6703@gmail.com! Logout

---

## Manage your account

### Change your account settings

[Profile](#)  
[Email](#)  
[Password](#)  
[Two-factor authentication](#)  
[Personal data](#)

#### Configure authenticator app

To use an authenticator app go through the following steps:

1. Download a two-factor authenticator app like Microsoft Authenticator for [Android](#) and [iOS](#) or Google Authenticator for [Android](#) and [iOS](#).
2. Scan the QR Code or enter this key `drqv zaij bxmd yvas nxai y7xq 5dsv 17ys` into your two factor authenticator app. Spaces and casing do not matter.

Learn how to enable QR code generation.

3. Once you have scanned the QR code or input the key above, your two factor authentication app will provide you with a unique code. Enter the code in the confirmation box below.

Verification Code

Verify

Рисунок 3.5 – Страница добавления двухфакторной аутентификации

### 3.4.3 Реализация бизнес-требований приложения

Для реализации бизнес требований приложения первым делом необходимо создать классы моделей и связать их через контекст базы данных

Entity Framework. Модели будем создавать в соответствии с подразделом 2.4 «Разработка структуры данных». Модели данных создадим в папке Models. Так как в базе данных присутствуют связи в виде внешних ключей, кроме непосредственного описания моделей необходимо задать связи между ними. Для этого используется интерфейс `IEntityTypeConfiguration`. Для задания маппинга нужно для каждой сущности создать наследника этого интерфейса. Конфигурация производится с помощью класса `EntityTypeBuilder`, который содержит следующие методы: `HasKey`, `HasAlternateKey`, `HasNoKey`, `HasData`, `HasIndex`, `Ignore`, `HasOne`, `HasMany` и другие. Зададим конфигурацию для каждой сущности. Внесем их в контекст базы данных, для этого добавим в него свойства типа `DbSet<T>`, где `T` – тип соответствующей сущности.

Теперь инкапсулируем логику доступа к данным в отдельных классах, используя паттерн репозиторий. Паттерн состоит в том, что любая логика доступа к базе данных, которую пользователь может вызвать, должна быть инкапсулирована в отдельном классе. Это позволяет эффективно отделить логику доступа к данным от бизнес логики, что, в свою очередь, сильно упрощает тестирование приложения. Схема паттерна изображена на рисунке 3.6. Реализуем репозитории для каждой сущности в базе данных. Примером реализации репозитория может служить репозиторий словарей. В нем реализуем следующие методы: `FindById`, `GetAll`, `FindByDictionaryGroup`, `Add`, `Delete`. Они позволяют осуществить типовые выборки и манипуляции над таблицей словарей.

Основную бизнес-логику отделим от контроллеров и сосредоточим в отдельных сервисах для облегчения тестирования приложения. Сервисы представляют из себя классы, которые инкапсулируют основные операции бизнес-логики и группируют их по смыслу.

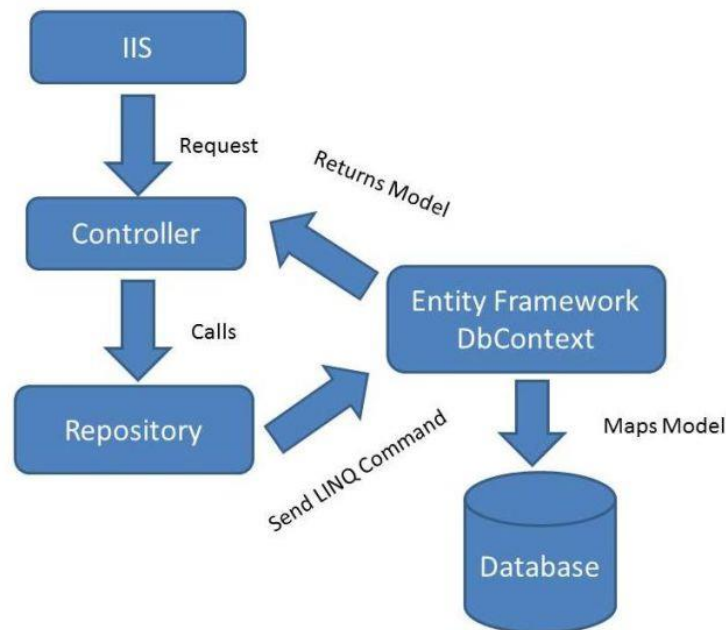


Рисунок 3.6 – Схема паттерна «Репозиторий»

Типовым сервисом можно считать `ExerciseCreationService`, который отвечает за создание набора упражнений по различным фильтрам: по списку выбранных упражнений, по выбранным группам словарей, словарям и по степени изученности лексики в словарях. Он, как и другие сервисы, может иметь зависимости от других сервисов, например от сервиса получения неправильных переводов `IncorrectTranslationsService`.

Все зависимости в приложении должны быть распространены через контейнер зависимостей, который предоставляет пакет “`Microsoft.Extensions.DependencyInjection`”. Все зависимости необходимо зарегистрировать в контейнере зависимостей в классе `Startup.cs` или, если сконфигурировано иначе, в указанном месте (в нашем случае в `InfrastructureInstaller`). В соответствии с принципом IoC (`InversionOfControl`) все зависимости быть только от интерфейсов, а не от конкретных реализаций. Таким образом, не только появляется возможность переключать реализации зависимых классов во время выполнения программы, но и увеличивается связность и уменьшается зацепление (что, в соответствии с принципами SOLID, улучшает качество программного кода и его поддерживаемость).

Пользуясь описанными выше принципами, реализуем домашнюю страницу, список группы словарей, список словарей внутри группы, страницу создания перевода, упражнения. На рисунке 3.7 изображена домашняя страница приложения. Она содержит общую статистику пользователя по изучению переводов во всех его словарях и список группы словарей. По нажатию на группу словарей, пользователю открывается список словарей, созданных внутри этой группы. В списке словарей можно выбрать словари для изучения. После выбора стоит нажать кнопку “Study” чтобы перейти на изучение.

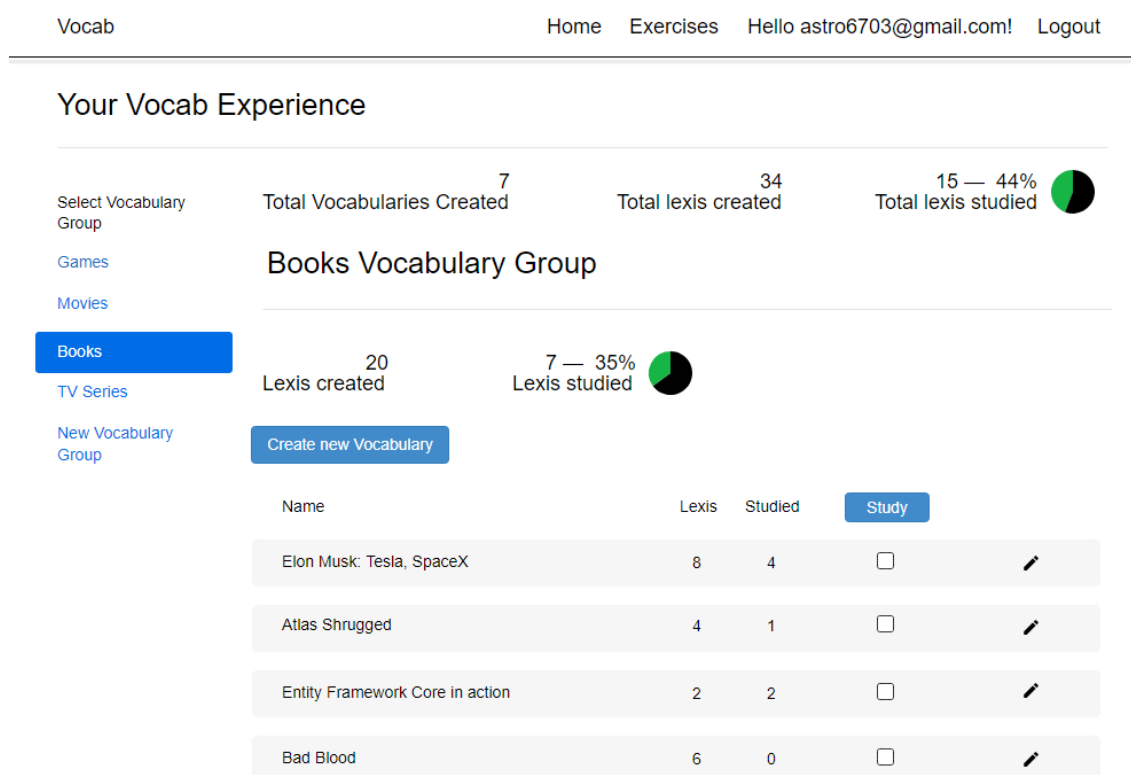


Рисунок 3.7 – Домашняя страница приложения

Страница «Словарь» содержит статистику по словарю и список лексики этого словаря в виде таблицы. Для каждой лексической единицы есть кнопка редактирования, отметка о запоминании и возможность выбора для упражнения. Присутствует ссылка назад на домашнюю страницу.

## Lexis of Vocabulary Elon Musk: Tesla, SpaceX

Lexis created 8

Lexis studied 4 — 50%


[Create new translation](#)

Name	Studied	Study	
dogged	✓	<input type="checkbox"/>	
meddle	✓	<input type="checkbox"/>	
peddle		<input type="checkbox"/>	
in due course	✓	<input type="checkbox"/>	
highfalutin		<input type="checkbox"/>	
beleaguered		<input type="checkbox"/>	
like chalk and cheese	✓	<input type="checkbox"/>	
anguish		<input type="checkbox"/>	

[Back to Vocabulary Groups](#)

Рисунок 3.8 – Страница «Словари»

На странице «упражнения» пользователь может пройти некоторые упражнения из следующего списка: определение, правилен ли данный перевод и выбор правильного перевода из списка. Остальные упражнения пока не были реализованы. На рисунках 3.9 и 3.10 изображены экраны разработанных упражнений.

Is the translation correct?

In order to finish the exercise, press the next button:

Finish

Anguish

лезть, вмешиваться

No

Yes

[Reveal context](#)

Рисунок 3.9 – Страница упражнения определения верности перевода

Select the correct translation

In order to finish the exercise, press the next button:

Finish

упрямый

meddle

anguish

peddle

dogged

like chalk and  
cheese

highfalutin

[Reveal context](#)

Рисунок 3.10 – Страница упражнения выбора правильного перевода

На рисунке 3.11 изображена страница создания перевода внутри словаря. Она содержит поля для ввода переводимой лексической единицы, ее списка переводов и ее контекстной информации. При вводе переводимой лексической единицы в списке переводов появляются доступные переводы, из которых пользователь может выбрать те, которых он хотел бы изучить. Также



при вводе переводимого слова в поле с контекстной информацией появляются определения данной лексики и примеры ее использования.

Vocab Home Exercises Hello astro6703@gmail.com! Logout

### Create the translation

Type in the word, colocation or sentence you would like to learn:

exaggerate

Select the translations you would like to remember:

утрировать

преувеличивать

File Edit View Insert Format Tools

↶ ↷ ↺ ↻

- It's no exaggeration to say that git represents a quantum leap in this area.
- Don't exaggerate! It wasn't that bad!
- **Greatly/grossly/wildly exaggerate something:** The paper's political influence has been greatly exaggerated.
- **Exaggerate the importance/significance of something:** We should not exaggerate the importance of this agreement.

[Back to Vocabulary](#) Save

Рисунок 3.11 – Страница создания перевода внутри словаря

### Выводы к разделу 3

В данном разделе был описан процесс разработки приложения на основе спроектированной в разделе 2 модели. Для этого были выбраны язык программирования C# и фреймворк ASP.NET Core из-за их зрелости, надежности и активного развития; были выбраны подходящие под требования электронный переводчик Yandex Translate и электронный словарь Macmillan Dictionary. Далее, были описаны этапы создания приложения и некоторые используемые в разработке паттерны проектирования.

В итоге, было разработано приложение в соответствии с требованиями и произведенным проектированием, которое дает пользователю возможность создать его личные словари, ассоциировать их с источниками на языке оригинала, в которых эта лексика была встречена и двумя видами упражнений.

## **ЗАКЛЮЧЕНИЕ**

В работе представлен процесс разработки информационной системы «Адаптивная система в помощь изучающему иностранный (английский) язык». Был выполнен полный цикл разработки программного обеспечения, включающий анализ существующих аналогов, формулирование требований, проектирование и реализацию функциональности приложения.

Таким образом, цель по разработке WEB-приложения для изучения английской лексики была достигнута.

Тем не менее, разработанное приложение не готово ко всеобщему пользованию. В дальнейшем следует реализовать более приятный дизайн и удобный в употреблении интерфейс пользователя. Кроме того, приложение отстает от аналогов по списку доступных упражнений, что следует исправить.

## ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

WEB (англ. web – паутина) – система доступа к связанным между собой документам на различных компьютерах, подключённых к Интернету,

DFD – общепринятое сокращение от англ. data flow diagrams – диаграммы потоков данных,

IDEF – I-CAM DEFinition или Integrated DEFinition – методологии семейства ICAM (Integrated Computer-Aided Manufacturing) для решения задач моделирования сложных систем,

BPMN – Business Process Management Notation – система условных обозначений (нотация) и их описания в XML для моделирования бизнес-процессов,

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных,

API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой,

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript,

SDK (от англ. software development kit) — набор средств разработки, позволяющий специалистам по программному обеспечению создавать приложения,

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»,

ПО – программное обеспечение,

SPA (англ. single page application) – это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX,

IoC (англ. Inversion of Control) — важный принцип объектно-ориентированного программирования, используемый для уменьшения зацепления (связанности) в компьютерных программах,

SOLID (сокр. от англ. single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion) в программировании — мнемонический акроним, введённый Майклом Фэзерсом (Michael Feathers) для первых пяти принципов, названных Робертом Мартином в начале 2000-х, которые означали 5 основных принципов объектно-ориентированного программирования и проектирования.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ethnologue (2021). What are the top 200 most spoken languages? [Электронный ресурс] // URL: <http://www.ethnologue.com/guides/ethnologue200> (дата обращения: 07.06.2021).
2. Майк Кордуэлл. Модель памяти Аткинсона-Шиффрина // Психология. А-Я. Словарь-справочник / Пер. с англ. К. С. Ткаченко. — ФАИР-ПРЕСС, 2000.
3. Е. Е. Васильева, В. Ю. Васильев. Суперпамять для всех. — М.: Аст, 2006. — 71 с. — (Суперпамять). — ISBN 5-17-038091-7.
4. Google Translate API [Электронный ресурс] // URL: <https://translation.googleapis.com/language> (дата обращения 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
5. Microsoft Text Translator API 3.0 [Электронный ресурс] // URL: <https://api.cognitive.microsofttranslator.com> (дата обращения 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
6. MyMemory API [Электронный ресурс] // URL: <https://api.mymemory.translated.net> (дата обращения 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
7. Translate.Yandex API [Электронный ресурс] // URL: <https://translate.api.cloud.yandex.net/translate/v2> (дата обращения 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
8. Oxford Dictionaries API [Электронный ресурс] // URL: <https://od-api.oxforddictionaries.com/api/v2> (дата обращения: 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
9. Macmillan Dictionary API [Электронный ресурс] // URL: <https://www.macmillandictionary.com/api/v1> (дата обращения: 03.04.2021). — Режим доступа: для зарегистрированных пользователей.
10. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. — СПб.: Питер, 2002. — 496 с.

11. Внедрение зависимостей в ASP.NET Core [Электронный ресурс] // URL: <https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/dependency-injection> (дата обращения: 04.04.2021).
12. Entity Framework Documentation [Электронный ресурс] // URL: <https://docs.microsoft.com/ef/> (дата обращения: 10.04.2021).
13. Введение в RazorPages в ASP.NET Core [Электронный ресурс] // URL: <https://docs.microsoft.com/aspnet/core/mvc/razor-pages> (дата обращения: 12.04.2021).
14. Представления в ASP.NET Core [Электронный ресурс] // URL: <https://docs.microsoft.com/aspnet/core/mvc/views/overview> (дата обращения: 12.04.2021).
15. Обработка запросов с помощью контроллеров в ASP.NET Core [Электронный ресурс] // URL: <https://docs.microsoft.com/ru-ru/aspnet/core/mvc/controllers/actions> (дата обращения: 13.04.2021).
16. Бишоп, Дж. С# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2011. - 472 с.

## ПРИЛОЖЕНИЕ А

### Программный код приложения

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Application;
using Infrastructure;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using WebAPI.Configurations;

namespace WebAPI
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCors(options =>
            {
                options.AddPolicy("AllowVocabApp",
                    builder => builder.WithOrigins("https://localhost:44321"));
            });

            services.AddJwtConfiguration(Configuration);

            services.AddApplication();

            services.AddInfrastructure(Configuration);

            services.AddControllers();
        }
    }
}
```

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseHttpsRedirection();

    app.UseRouting();

    app.UseCors("AllowVocabApp");

    app.UseAuthorization();

    app.UseAuthentication();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

```
public class Program
```

```
{
    public static async Task Main(string[] args)
    {
        var host = CreateWebHostBuilder(args).Build();

        using (var serviceScope = host.Services.CreateScope())
        {
            var roleManager =
                serviceScope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

            if (!await roleManager.RoleExistsAsync("User"))
            {
                var userRole = new IdentityRole("User");
                await roleManager.CreateAsync(userRole);
            }
        }

        await host.RunAsync();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
```



```

        webBuilder.UseStartup<Startup>();
    });

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args).UseStartup<Startup>();
}

public static class JwtConfiguration
{
    public static IServiceCollection AddJwtConfiguration(this IServiceCollection services,
        IConfiguration configuration)
    {
        //For Jwt Configuration And Authentication Configuration-----
        -----

        var jwtSettings = new JwtSettings();
        configuration.Bind(nameof(jwtSettings), jwtSettings);
        services.AddSingleton(jwtSettings);

        var tokenValidationParameters =
            new TokenValidationParameters()
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes((jwtSettings.Secret))),
                ValidateIssuer = false,
                ValidateAudience = false,
                RequireExpirationTime = false,
                ValidateLifetime = true
            };
        services.AddSingleton(tokenValidationParameters);

        services.AddAuthentication(options =>
        {
            options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        })
        .AddJwtBearer(options =>
        {
            options.SaveToken = true;
            options.TokenValidationParameters = tokenValidationParameters;
        });

        //For Jwt configuration End-----
        -----

        services.AddAuthorization();

        return services;
    }
}

```

```

public static class InfrastructureInstaller
{
    public static IServiceCollection AddInfrastructure(this IServiceCollection services,
        IConfiguration configuration)
    {
        services.AddDbContext<VocabularyBuilderDbContext>(options =>
            options.UseSqlServer(
                configuration.GetConnectionString("VocabDatabase"),
                b
b.MigrationsAssembly(typeof(VocabularyBuilderDbContext).Assembly.FullName))); =>

        services.AddScoped<IVocabularyBuilderDbContext>(provider
provider.GetService<VocabularyBuilderDbContext>()); =>

        services.AddDefaultIdentity<ApplicationUser>()
            .AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<VocabularyBuilderDbContext>();

        services.AddTransient<IIdentityService, IdentityService>();
        services.AddScoped<ICurrentUserService, CurrentUserService>();

        return services;
    }
}

public class IdentityService: IIdentityService
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;
    private readonly JwtSettings _jwtSettings;
    private readonly TokenValidationParameters _tokenValidationParameters;
    private readonly IVocabularyBuilderDbContext _context;

    public IdentityService(UserManager<ApplicationUser> userManager,
        RoleManager<IdentityRole> roleManager, JwtSettings jwtSettings, TokenValidationParameters
        tokenValidationParameters, IVocabularyBuilderDbContext context)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _jwtSettings = jwtSettings;
        _tokenValidationParameters = tokenValidationParameters;
        _context = context;
    }

    public async Task<AuthenticationResult> RegisterAsync(string email, string password,
        UserRole? role)
    {
        var existingUser = await _userManager.FindByEmailAsync(email);

        if (existingUser != null)
        {
            return new AuthenticationResult
            {

```

```

        Errors = new[] { "User with this email address already exists" }
    };
}

var newUserId = Guid.NewGuid();
var newUser = new ApplicationUser
{
    Id = newUserId.ToString(),
    Email = email,
    UserName = email
};

var createdUser = await _userManager.CreateAsync(newUser, password);

if (!createdUser.Succeeded)
{
    return new AuthenticationResult
    {
        Errors = createdUser.Errors.Select(x => x.Description)
    };
}

if (role.HasValue)
    await AssignRoleToUser(role.Value, newUser);

//await _userManager.AddClaimAsync(newUser, new Claim("User.view", "true"));

return await GenerateAuthenticationResultForUserAsync(newUser);
}

public async Task<AuthenticationResult> LoginAsync(string email, string password)
{
    var user = await _userManager.FindByEmailAsync(email);

    if (user == null)
    {
        return new AuthenticationResult
        {
            Errors = new[] { "User does not exist." }
        };
    }

    var userHasValidPassword = await _userManager.CheckPasswordAsync(user, password);

    if (!userHasValidPassword)
    {
        return new AuthenticationResult
        {
            Errors = new[] { "User/password combination is wrong" }
        };
    }
}

```

```

    return await GenerateAuthenticationResultForUserAsync(user);
}

public async Task<AuthenticationResult> RefreshTokenAsync(string token, string
refreshToken)
{
    var validatedToken = GetPrincipalFromToken(token);

    if (validatedToken == null)
    {
        return new AuthenticationResult()
        {
            Errors = new[] { "Invalid Token" }
        };
    }

    var expiryDateUnix =
        long.Parse(validatedToken.Claims.Single(x => x.Type ==
JwtRegisteredClaimNames.Exp).Value);

    var expiryDateTimeUtc = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)
        .AddSeconds(expiryDateUnix);

    if (expiryDateTimeUtc > DateTime.UtcNow)
    {
        return new AuthenticationResult()
        {
            Errors = new[] { "This token hasn't expired yet" }
        };
    }
    var jti = validatedToken.Claims.Single(x => x.Type ==
JwtRegisteredClaimNames.Jti).Value;

    var storeRefreshToken = await _context.RefreshTokens.SingleOrDefaultAsync(x =>
x.Token == refreshToken);

    if (storeRefreshToken == null)
    {
        return new AuthenticationResult
        {
            Errors = new[] { "This refresh token doesn't exist" }
        };
    }

    if (DateTime.UtcNow > storeRefreshToken.ExpiryDate)
    {
        return new AuthenticationResult() { Errors = new[] { "This refresh token has expired" }
    };
    }

    if (storeRefreshToken.Invalidated)

```

```

        {
            return new AuthenticationResult() { Errors = new[] { "This refresh token has been
invalidated" } };
        }

        if (storeRefreshToken.Used)
        {
            return new AuthenticationResult() { Errors = new[] { "This refresh token has been used"
} };
        }

        if (storeRefreshToken.JwtId != jti)
        {
            return new AuthenticationResult() { Errors = new[] { "This refresh token does not match
this Jwt" } };
        }

        storeRefreshToken.Used = true;
        _context.RefreshTokens.Update(storeRefreshToken);
        await _context.SaveChangesAsync( new CancellationToken());

        var user = await _userManager.FindByIdAsync(validatedToken.Claims.Single(x =>
x.Type == "id").Value);

        return await GenerateAuthenticationResultForUserAsync(user);
    }

    public async Task<string> GetUserNameAsync(string userId)
    {
        var user = await _userManager.Users.FirstAsync(u => u.Id == userId);

        return user.UserName;
    }

    private async Task<AuthenticationResult>
GenerateAuthenticationResultForUserAsync(ApplicationUser user)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var key = Encoding.ASCII.GetBytes(_jwtSettings.Secret);

        var claims = new List<Claim>
        {
            new Claim(JwtRegisteredClaimNames.Sub, user.Email),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            new Claim(JwtRegisteredClaimNames.Email, user.Email),
            new Claim("id", user.Id)
        };

        //In case we add policy to user by claim
        var userClaims = await _userManager.GetClaimsAsync(user);
        claims.AddRange(userClaims);
    }

```

```

//Adding roleClaim of User in jwt claims
var userRoles = await _userManager.GetRolesAsync(user);
foreach (var userRole in userRoles)
{
    claims.Add(new Claim(ClaimTypes.Role, userRole));
    var role = await _roleManager.FindByNameAsync(userRole);
    if (role == null) continue;
    var roleClaims = await _roleManager.GetClaimsAsync(role);
    foreach (var roleClaim in roleClaims)
    {
        if (claims.Contains(roleClaim))
            continue;

        claims.Add(roleClaim);
    }
}

var tokenDescriptor = new SecurityTokenDescriptor
{
    Subject = new ClaimsIdentity(claims),
    Expires = DateTime.UtcNow.Add(_jwtSettings.TokenLifetime),
    SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
};

var token = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken = new RefreshToken()
{
    JwtId = token.Id,
    UserId = user.Id,
    CreationDate = DateTime.UtcNow,
    ExpiryDate = DateTime.UtcNow.AddMonths(6),
};

await _context.RefreshTokens.AddAsync(refreshToken);
await _context.SaveChangesAsync(new CancellationToken());

return new AuthenticationResult
{
    Success = true,
    Token = tokenHandler.WriteToken(token),
    RefreshToken = refreshToken.Token
};
}

private ClaimsPrincipal GetPrincipalFromToken(string token)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    try

```

```

    {
        var tokenValidationParameters = _tokenValidationParameters.Clone();
        tokenValidationParameters.ValidateLifetime = false;
        var principal = tokenHandler.ValidateToken(token, tokenValidationParameters, out var
validatedToken);

        if (!IsJwtWithValidSecurityAlgorithm(validatedToken))
        {
            return null;
        }

        return principal;
    }
    catch
    {
        return null;
    }
}

private bool IsJwtWithValidSecurityAlgorithm(SecurityToken validatedToken)
{
    return (validatedToken is JwtSecurityToken jwtSecurityToken) &&
        jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
            StringComparison.InvariantCultureIgnoreCase);
}

private async Task AssignRoleToUser(UserRole role, ApplicationUser user)
{
    switch (role)
    {
        case UserRole.Admin:
            await _userManager.AddToRoleAsync(user, "Admin");
            break;
        case UserRole.User:
            await _userManager.AddToRoleAsync(user, "User");
            break;
    }
}

public static class GeneralExtensions
{
    public static string GetUserId(this HttpContext httpContext)
    {
        if (httpContext == null)
        {
            return string.Empty;
        }
        return httpContext.User.Claims.Single(x => x.Type == "id").Value;
    }
}

```

```

public class CurrentUserService:ICurrentUserService
{
    public string UserId { get; }
    public CurrentUserService(IHttpContextAccessor httpContextAccessor)
    {
        UserId = httpContextAccessor.HttpContext?.User?.Claims?.Single(u => u.Type ==
"id").Value;
    }
}

public class ApplicationUser:IdentityUser
{
}

public class LexisEntry {
    public int LexisEntryId { get; set; }

    public string Entry { get; set; }

    public IEnumerable<Translation> Translations { get; set; }

    public string Context { get; set; }

    public bool IsStudied { get; set; }

    public byte[] ContextContent { get; set; }

    public int VocabularyId { get; set; }
}

public class Translation
{
    public int LexisEntryId { get; set; }
    public string Text { get; set; }
    public string Example { get; set; }
    public int ImportedWordId { get; set; }
}

public class VocabularyGroup {
    public int VocabularyGroupId { get; set; }

    public string Name { get; set; }

    public string Description { get; set; }

    public IEnumerable<Vocabulary> Vocabularies { get; set; }
}

public class Vocabulary {
    public int VocabularyId { get; set; }

    public string Name { get; set; }
}

```



```

    public string Description { get; set; }

    public int VocabularyGroupId { get; set; }

    public IEnumerable<LexisEntry> Entries { get; set; }
}

[DataContract]
public class FileUploadModel
{
    [DataMember(Name = "fileName")]
    public string FileName { get; set; }

    [DataMember(Name = "fileBytes")]
    public byte[] FileBytes { get; set; }
}

public class RefreshToken
{
    public string Token { get; set; }
    public string JwtId { get; set; }
    public DateTime CreationDate { get; set; }
    public DateTime ExpiryDate { get; set; }
    public bool Used { get; set; }
    public bool Invalidated { get; set; }
    public string UserId { get; set; }
}

public class ValidationBehavior<TRequest, TResponse> : IPipelineBehavior<TRequest,
TResponse>
    where TRequest : IRequest<TResponse>
{
    private readonly IEnumerable<IValidator<TRequest>> _validators;

    public ValidationBehavior(IEnumerable<IValidator<TRequest>> validators)
    {
        _validators = validators;
    }

    public async Task<TResponse> Handle(TRequest request, CancellationToken
cancellationTokn, RequestHandlerDelegate<TResponse> next)
    {
        if (_validators.Any())
        {
            var context = new ValidationContext(request);

            var validationResults = await Task.WhenAll(_validators.Select(v =>
v.ValidateAsync(context, cancellationTokn)));
            var failures = validationResults.SelectMany(r => r.Errors).Where(f => f !=
null).ToList();

```

```

        if (failures.Count != 0)
            throw new ValidationException(failures);
    }
    return await next();
}
}

public class UnhandledExceptionBehaviour<TRequest, TResponse> :
    IPipelineBehavior<TRequest, TResponse>
{
    private readonly ILogger<TRequest> _logger;

    public UnhandledExceptionBehaviour(ILogger<TRequest> logger)
    {
        _logger = logger;
    }

    public async Task<TResponse> Handle(TRequest request, CancellationToken
    cancellationToken, RequestHandlerDelegate<TResponse> next)
    {
        try
        {
            return await next();
        }
        catch (Exception ex)
        {
            var requestName = typeof(TRequest).Name;

            _logger.LogError(ex, "CleanArchitecture Request: Unhandled Exception for Request
            {Name} { @Request}", requestName, request);

            throw;
        }
    }
}

public class PerformanceBehaviour<TRequest, TResponse> : IPipelineBehavior<TRequest,
TResponse>
{
    private readonly Stopwatch _timer;
    private readonly ILogger<TRequest> _logger;
    private readonly ICurrentUserService _currentUserService;
    private readonly IIdentityService _identityService;

    public PerformanceBehaviour(
        ILogger<TRequest> logger,
        ICurrentUserService currentUserService,
        IIdentityService identityService)
    {
        _timer = new Stopwatch();

        _logger = logger;
    }
}

```

```

        _currentUserService = currentUserService;
        _identityService = identityService;
    }

    public async Task<TResponse> Handle(TRequest request, CancellationToken
cancellationTokentoken, RequestHandlerDelegate<TResponse> next)
    {
        _timer.Start();

        var response = await next();

        _timer.Stop();

        var elapsedMilliseconds = _timer.ElapsedMilliseconds;

        if (elapsedMilliseconds > 500)
        {
            var requestName = typeof(TRequest).Name;
            var userId = _currentUserService.UserId ?? string.Empty;
            var userName = string.Empty;

            if (!string.IsNullOrEmpty(userId))
            {
                userName = await _identityService.GetUserNameAsync(userId);
            }

            _logger.LogWarning("CleanArchitecture Long Running Request: {Name}
({ElapsedMilliseconds} milliseconds) { @UserId } { @UserName } { @Request}",
                requestName, elapsedMilliseconds, userId, userName, request);
        }

        return response;
    }
}

public class LoggingBehaviour<TRequest> : IRequestPreProcessor<TRequest>
{
    private readonly ILogger<TRequest> _logger;
    private readonly ICurrentUserService _currentUserService;
    private readonly IIdentityService _identityService;

    public LoggingBehaviour(ILogger<TRequest> logger, ICurrentUserService
currentUserService, IIdentityService identityService)
    {
        _logger = logger;
        _currentUserService = currentUserService;
        _identityService = identityService;
    }

    public async Task Process(TRequest request, CancellationToken cancellationTokentoken)
    {
        var requestName = typeof(TRequest).Name;

```

```

var userId = _currentUserService.UserId ?? string.Empty;
string userName = string.Empty;

if (!string.IsNullOrEmpty(userId))
{
    userName = await _identityService.GetUserNameAsync(userId);
}

_logger.LogInformation("CleanArchitecture      Request:      {Name}      {@UserId}
{@UserName} {@Request}",
    requestName, userId, userName, request);
}
}

public class VocabularyGroupDto : IMapFrom<VocabularyGroupDto>
{
    public int Id { get; set; }
    public string Description { get; set; }

    public void Mapping(Profile profile)
    {
        profile.CreateMap<Category, VocabularyGroupDto>()
            .ForMember(d => d.Description, opt => opt.MapFrom(e => e.Description))
            .ForMember(d => d.Id, opt => opt.MapFrom(e => e.CategoryId));
    }
}

public class VocabularyGroup : IMapFrom<VocabularyGroup>
{
    public int Id { get; set; }
    public string Description { get; set; }

    public void Mapping(Profile profile)
    {
        profile.CreateMap<VocabularyGroup, VocabularyGroup>()
            .ForMember(d => d.Description, opt => opt.MapFrom(e => e.Description))
            .ForMember(d => d.Id, opt => opt.MapFrom(e => e.Id));
    }
}

public class AuthenticationResult
{
    public string Token { get; set; }

    public string RefreshToken { get; set; }
    public bool Success { get; set; }
    public IEnumerable<string> Errors { get; set; }
}

public interface IVocabularyBuilderDbContext
{
    DbSet<VocabularyGroup> Categories { get; set; }
}

```

```

    DbSet<Vocabulary> TypeCards { get; set; }
    DbSet<LexisEntry> FlashCards { get; set; }
    DbSet<Translation> RefreshTokens { get; set; }

    Task<int> SaveChangesAsync(Cancellation_token cancellationToken);
}

public class ValidationException : Exception
{
    public ValidationException()
        : base("One or more validation failures have occurred.")
    {
        Errors = new Dictionary<string, string[]>();
    }

    public ValidationException(IEnumerable<ValidationFailure> failures)
        : this()
    {
        var failureGroups = failures
            .GroupBy(e => e.PropertyName, e => e.ErrorMessage);

        foreach (var failureGroup in failureGroups)
        {
            var propertyName = failureGroup.Key;
            var propertyFailures = failureGroup.ToArray();

            Errors.Add(propertyName, propertyFailures);
        }
    }

    public IDictionary<string, string[]> Errors { get; }
}

public class VocabularyRepository : IVocabularyRepository<VocabularyEntity>
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly VocabularyQueries _vocabularyQueries;
    public VocabularyRepository(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
        _vocabularyQueries = new VocabularyQueries();
    }

    public async Task<IEnumerable<VocabularyEntity>> GetAllAsync()
    {
        IEnumerable<VocabularyEntity> vocabularies;
        vocabularies =
        await _unitOfWork.Connection.QueryAsync<VocabularyEntity>(_vocabularyQueries.SelectAllQuery
    );
        return vocabularies;
    }
}

```

```

    public async Task<IEnumerable<VocabularyEntity>> GetManyAsync(object filter)
    {
        IEnumerable<VocabularyEntity> vocabularies;
        string sql = _vocabularyQueries.SelectManyQuery +
QueryBuilder.PrepareCondition(filter);
        vocabularies = await _unitOfWork.Connection.QueryAsync<VocabularyEntity>(sql);
        return vocabularies;
    }

    public async Task<int> AddAsync(VocabularyEntity entity)
    {
        return await _unitOfWork.Connection.ExecuteAsync(_vocabularyQueries.InsertQuery,
entity);
    }

    public async Task UpdateAsync(VocabularyEntity entity)
    {
        await _unitOfWork.Connection.ExecuteAsync(_vocabularyQueries.UpdateQuery, entity);
    }

    public async Task DeleteAsync(int id)
    {
        await _unitOfWork.Connection.ExecuteAsync(_vocabularyQueries.DeleteQuery, new { Id
= id});
    }
}

await _unitOfWork.CommitAsync();
}

public async Task DeleteVocabulary(Vocabulary vocabulary)
{
    _unitOfWork.Vocabularies.Remove(vocabulary);

    await _unitOfWork.CommitAsync();
}
}

const baseUrl = document.getElementsByTagName('base')[0].getAttribute('href');
const rootElement = document.getElementById('root');

ReactDOM.render(
    <BrowserRouter basename={baseUrl}>
        <App />
    </BrowserRouter>,
    rootElement);

import './custom.css'

```

```

export default class App extends Component {
  static displayName = App.name;

  render () {
    return (
      <Layout>
        <Route exact path="/" component={Home} />
        <Route exact path='home/' component={Home} />
        <Route path='exercise/' component={Exercise} />
        <AuthorizeRoute path='/vocabularies' component={Vocabularies} />
        <Route
          path={ApplicationPaths.ApiAuthorizationPrefix}
component={ApiAuthorizationRoutes} />
      </Layout>
    );
  }
}

const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
  window.location.hostname === '[:1]' ||
  window.location.hostname.match(
    /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)?)?{3}$/
  )
);

export default function register () {
  if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator) {
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location);
    if (publicUrl.origin !== window.location.origin) {
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        checkValidServiceWorker(swUrl);
      } else {
        registerValidSW(swUrl);
      }
    });
  }
}

function registerValidSW (swUrl) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
      registration.onupdatefound = () => {
        const installingWorker = registration.installing;
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {

```

```

    if (navigator.serviceWorker.controller) {
      console.log('New content is available; please refresh.');
```

```
    } else {
      console.log('Content is cached for offline use.');
```

```
    }
  }
};
});
})
.catch(error => {
  console.error('Error during service worker registration:', error);
});
}

function checkValidServiceWorker (swUrl) {
  fetch(swUrl)
    .then(response => {
      if (
        response.status === 404 ||
        response.headers.get('content-type').indexOf('javascript') === -1
      ) {
        navigator.serviceWorker.ready.then(registration => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        registerValidSW(swUrl);
      }
    })
    .catch(() => {
      console.log(
        'No internet connection found. App is running in offline mode.'
      );
    });
  }

export function unregister () {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready.then(registration => {
      registration.unregister();
    });
  }
}

export class Layout extends Component {
  static displayName = Layout.name;

  render () {
    return (
      <div>
        <NavMenu />

```



```

    <Container>
      {this.props.children}
    </Container>
  </div>
);
}
}

export class NavMenu extends Component {
  static displayName = NavMenu.name;

  constructor (props) {
    super(props);

    this.toggleNavbar = this.toggleNavbar.bind(this);
    this.state = {
      collapsed: true
    };
  }

  toggleNavbar () {
    this.setState({
      collapsed: !this.state.collapsed
    });
  }

  render () {
    return (
      <header>
        <Navbar className="navbar-expand-sm navbar-toggleable-sm ng-white border-bottom box-shadow mb-3" light>
          <Container>
            <NavbarBrand tag={Link} to="/">Vocab</NavbarBrand>
            <NavbarToggler onClick={this.toggleNavbar} className="mr-2" />
            <Collapse
              className="d-sm-inline-flex"
              flex-sm-row-reverse"
              isOpen={!this.state.collapsed} navbar>
              <ul className="navbar-nav flex-grow">
                <NavItem>
                  <NavLink tag={Link} className="text-dark" to="/">Home</NavLink>
                </NavItem>
                <NavItem>
                  <NavLink tag={Link} className="text-dark" to="/counter">Exercise</NavLink>
                </NavItem>
                <LoginMenu>
                </LoginMenu>
              </ul>
            </Collapse>
          </Container>
        </Navbar>
      </header>
    );
  }
}

```

```

}export const ApplicationName = 'ASP.NETCoreWebApplication';

export const QueryParameterNames = {
  returnUrl: 'returnUrl',
  Message: 'message'
};

export const LogoutActions = {
  LogoutCallback: 'logout-callback',
  Logout: 'logout',
  LoggedOut: 'logged-out'
};

export const LoginActions = {
  Login: 'login',
  LoginCallback: 'login-callback',
  LoginFailed: 'login-failed',
  Profile: 'profile',
  Register: 'register'
};

const prefix = '/authentication';

export const ApplicationPaths = {
  DefaultLoginRedirectPath: '/',
  ApiAuthorizationClientConfigurationUrl: `_${configuration}/${ApplicationName}`,
  ApiAuthorizationPrefix: prefix,
  Login: `${prefix}/${LoginActions.Login}`,
  LoginFailed: `${prefix}/${LoginActions.LoginFailed}`,
  LoginCallback: `${prefix}/${LoginActions.LoginCallback}`,
  Register: `${prefix}/${LoginActions.Register}`,
  Profile: `${prefix}/${LoginActions.Profile}`,
  LogOut: `${prefix}/${LogoutActions.Logout}`,
  LoggedOut: `${prefix}/${LogoutActions.LoggedOut}`,
  LogOutCallback: `${prefix}/${LogoutActions.LogoutCallback}`,
  IdentityRegisterPath: 'Identity/Account/Register',
  IdentityManagePath: 'Identity/Account/Manage'
};

import React, { Component, Fragment } from 'react';
import { Route } from 'react-router';
import { Login } from './Login'
import { Logout } from './Logout'
import { ApplicationPaths, LoginActions, LogoutActions } from './ApiAuthorizationConstants';

export default class ApiAuthorizationRoutes extends Component {

  render () {
    return(
      <Fragment>
        <Route path={ApplicationPaths.Login} render={() => loginAction(LoginActions.Login)}
      />

```

```

        <Route          path={ ApplicationPaths.LoginFailed}          render={()          =>
loginAction(LoginActions.LoginFailed)} />
        <Route          path={ ApplicationPaths.LoginCallback}          render={()          =>
loginAction(LoginActions.LoginCallback)} />
        <Route          path={ ApplicationPaths.Profile}          render={()          =>
loginAction(LoginActions.Profile)} />
        <Route          path={ ApplicationPaths.Register}          render={()          =>
loginAction(LoginActions.Register)} />
        <Route          path={ ApplicationPaths.LogOut}          render={()          =>
logoutAction(LogoutActions.Logout)} />
        <Route          path={ ApplicationPaths.LogOutCallback}          render={()          =>
logoutAction(LogoutActions.LogoutCallback)} />
        <Route          path={ ApplicationPaths.LoggedOut}          render={()          =>
logoutAction(LogoutActions.LoggedOut)} />
    </Fragment>;
}
}

function loginAction(name){
    return (<Login action={name}></Login>);
}

function logoutAction(name) {
    return (<Logout action={name}></Logout>);
}

import React from 'react'
import { Component } from 'react'
import { Route, Redirect } from 'react-router-dom'
import { ApplicationPaths, QueryParameterNames } from './ApiAuthorizationConstants'
import authService from './AuthorizeService'

export default class AuthorizeRoute extends Component {
    constructor(props) {
        super(props);

        this.state = {
            ready: false,
            authenticated: false
        };
    }

    componentDidMount() {
        this._subscription = authService.subscribe(() => this.authenticationChanged());
        this.populateAuthenticationState();
    }

    componentWillUnmount() {
        authService.unsubscribe(this._subscription);
    }

    render() {

```

```

    const { ready, authenticated } = this.state;
    var link = document.createElement("a");
    link.href = this.props.path;
    const                                returnUrl                                =
` ${link.protocol}://${link.host}${link.pathname}${link.search}${link.hash}`;
    const                                redirectUrl                                =
` ${ApplicationPaths.Login}?${QueryParameterNames.ReturnUrl}=${encodeURIComponent(redirectUrl)}`

    if (!ready) {
        return <div></div>;
    } else {
        const { component: Component, ...rest } = this.props;
        return <Route {...rest}
            render={ (props) => {
                if (authenticated) {
                    return <Component {...props} />
                } else {
                    return <Redirect to={redirectUrl} />
                }
            } } />
    }
}

async populateAuthenticationState() {
    const authenticated = await authService.isAuthenticated();
    this.setState({ ready: true, authenticated });
}

async authenticationChanged() {
    this.setState({ ready: false, authenticated: false });
    await this.populateAuthenticationState();
}

}

export class AuthorizeService {
    _callbacks = [];
    _nextSubscriptionId = 0;
    _user = null;
    _isAuthenticated = false;

    // By default pop ups are disabled because they don't work properly on Edge.
    // If you want to enable pop up authentication simply set this flag to false.
    _popUpDisabled = true;

    async isAuthenticated() {
        const user = await this.getUser();
        return !!user;
    }

    async getUser() {
        if (this._user && this._user.profile) {
            return this._user.profile;
        }
    }
}

```

```

    }

    await this.ensureUserManagerInitialized();
    const user = await this.userManager.getUser();
    return user && user.profile;
  }

  async getAccessToken() {
    await this.ensureUserManagerInitialized();
    const user = await this.userManager.getUser();
    return user && user.access_token;
  }

  // We try to authenticate the user in three different ways:
  // 1) We try to see if we can authenticate the user silently. This happens
  //    when the user is already logged in on the IdP and is done using a hidden iframe
  //    on the client.
  // 2) We try to authenticate the user using a PopUp Window. This might fail if there is a
  //    Pop-Up blocker or the user has disabled PopUps.
  // 3) If the two methods above fail, we redirect the browser to the IdP to perform a traditional
  //    redirect flow.
  async signIn(state) {
    await this.ensureUserManagerInitialized();
    try {
      const silentUser = await this.userManager.signInSilent(this.createArguments());
      this.updateState(silentUser);
      return this.success(state);
    } catch (silentError) {
      // User might not be authenticated, fallback to popup authentication
      console.log("Silent authentication error: ", silentError);

      try {
        if (this._popUpDisabled) {
          throw new Error('Popup disabled. Change
\AuthorizeService.js:AuthorizeService._popupDisabled\' to false to enable it.')
        }

        const popUpUser = await this.userManager.signInPopup(this.createArguments());
        this.updateState(popUpUser);
        return this.success(state);
      } catch (popUpError) {
        if (popUpError.message === "Popup window closed") {
          // The user explicitly cancelled the login action by closing an opened popup.
          return this.error("The user closed the window.");
        } else if (!this._popUpDisabled) {
          console.log("Popup authentication error: ", popUpError);
        }
      }

      // PopUps might be blocked by the user, fallback to redirect
      try {
        await this.userManager.signInRedirect(this.createArguments(state));
        return this.redirect();
      }
    }
  }

```

```

    } catch (redirectError) {
      console.log("Redirect authentication error: ", redirectError);
      return this.error(redirectError);
    }
  }
}

async completeSignIn(url) {
  try {
    await this.ensureUserManagerInitialized();
    const user = await this.userManager.signInCallback(url);
    this.updateState(user);
    return this.success(user && user.state);
  } catch (error) {
    console.log("There was an error signing in: ", error);
    return this.error("There was an error signing in.");
  }
}

// We try to sign out the user in two different ways:
// 1) We try to do a sign-out using a PopUp Window. This might fail if there is a
//    Pop-Up blocker or the user has disabled PopUps.
// 2) If the method above fails, we redirect the browser to the IdP to perform a traditional
//    post logout redirect flow.
async signOut(state) {
  await this.ensureUserManagerInitialized();
  try {
    if (this._popupDisabled) {
      throw new Error('Popup disabled. Change
\AuthorizeService.js:AuthorizeService._popupDisabled\' to false to enable it.')
    }

    await this.userManager.signoutPopup(this.createArguments());
    this.updateState(undefined);
    return this.success(state);
  } catch (popupSignOutError) {
    console.log("Popup signout error: ", popupSignOutError);
    try {
      await this.userManager.signoutRedirect(this.createArguments(state));
      return this.redirect();
    } catch (redirectSignOutError) {
      console.log("Redirect signout error: ", redirectSignOutError);
      return this.error(redirectSignOutError);
    }
  }
}

async completeSignOut(url) {
  await this.ensureUserManagerInitialized();
  try {
    const response = await this.userManager.signoutCallback(url);

```

```

        this.updateState(null);
        return this.success(response && response.data);
    } catch (error) {
        console.log(`There was an error trying to log out '${error}'.`);
        return this.error(error);
    }
}

updateState(user) {
    this._user = user;
    this._isAuthenticated = !!this._user;
    this.notifySubscribers();
}

subscribe(callback) {
    this._callbacks.push({ callback, subscription: this._nextSubscriptionId++ });
    return this._nextSubscriptionId - 1;
}

unsubscribe(subscriptionId) {
    const subscriptionIndex = this._callbacks
        .map((element, index) => element.subscription === subscriptionId ? { found: true, index
    } : { found: false })
        .filter(element => element.found === true);
    if (subscriptionIndex.length !== 1) {
        throw new Error(`Found an invalid number of subscriptions
    ${subscriptionIndex.length}`);
    }

    this._callbacks.splice(subscriptionIndex[0].index, 1);
}

notifySubscribers() {
    for (let i = 0; i < this._callbacks.length; i++) {
        const callback = this._callbacks[i].callback;
        callback();
    }
}

createArguments(state) {
    return { useReplaceToNavigate: true, data: state };
}

error(message) {
    return { status: AuthenticationResultStatus.Fail, message };
}

success(state) {
    return { status: AuthenticationResultStatus.Success, state };
}

redirect() {

```

```

    return { status: AuthenticationResultStatus.Redirect };
  }

  async ensureUserManagerInitialized() {
    if (this.userManager !== undefined) {
      return;
    }

    let response = await fetch(ApplicationPaths.ApiAuthorizationClientConfigurationUrl);
    if (!response.ok) {
      throw new Error(` Could not load settings for '${ApplicationName}'`);
    }

    let settings = await response.json();
    settings.automaticSilentRenew = true;
    settings.includeIdTokenInSilentRenew = true;
    settings.userStore = new WebStorageStateStore({
      prefix: ApplicationName
    });

    this.userManager = new UserManager(settings);

    this.userManager.events.addUserSignedOut(async () => {
      await this.userManager.removeUser();
      this.updateState(undefined);
    });
  }

  static get instance() { return authService }
}

const authService = new AuthorizeService();

export default authService;

export const AuthenticationResultStatus = {
  Redirect: 'redirect',
  Success: 'success',
  Fail: 'fail'
};

export class Login extends Component {
  constructor(props) {
    super(props);

    this.state = {
      message: undefined
    };
  }

  componentDidMount() {
    const action = this.props.action;
  }
}

```



```

switch (action) {
  case LoginActions.Login:
    this.login(this.getReturnUrl());
    break;
  case LoginActions.LoginCallback:
    this.processLoginCallback();
    break;
  case LoginActions.LoginFailed:
    const params = new URLSearchParams(window.location.search);
    const error = params.get(QueryParameterNames.Message);
    this.setState({ message: error });
    break;
  case LoginActions.Profile:
    this.redirectToProfile();
    break;
  case LoginActions.Register:
    this.redirectToRegister();
    break;
  default:
    throw new Error(`Invalid action '${action}'`);
}
}

render() {
  const action = this.props.action;
  const { message } = this.state;

  if (!!message) {
    return <div>{message}</div>
  } else {
    switch (action) {
      case LoginActions.Login:
        return (<div>Processing login</div>);
      case LoginActions.LoginCallback:
        return (<div>Processing login callback</div>);
      case LoginActions.Profile:
      case LoginActions.Register:
        return (<div></div>);
      default:
        throw new Error(`Invalid action '${action}'`);
    }
  }
}

async login(returnUrl) {
  const state = { returnUrl };
  const result = await authService.signIn(state);
  switch (result.status) {
    case AuthenticationResultStatus.Redirect:
      break;
    case AuthenticationResultStatus.Success:
      await this.navigateToReturnUrl(returnUrl);

```

```

        break;
    case AuthenticationResultStatus.Fail:
        this.setState({ message: result.message });
        break;
    default:
        throw new Error(`Invalid status result ${result.status}.`);
    }
}

async processLoginCallback() {
    const url = window.location.href;
    const result = await authService.completeSignIn(url);
    switch (result.status) {
        case AuthenticationResultStatus.Redirect:
            // There should not be any redirects as the only time completeSignIn finishes
            // is when we are doing a redirect sign in flow.
            throw new Error('Should not redirect.');
```

case AuthenticationResultStatus.Success:

```

            await this.navigateToReturnUrl(this.getReturnUrl(result.state));
            break;
        case AuthenticationResultStatus.Fail:
            this.setState({ message: result.message });
            break;
        default:
            throw new Error(`Invalid authentication result status '${result.status}'.`);
    }
}

getReturnUrl(state) {
    const params = new URLSearchParams(window.location.search);
    const fromQuery = params.get(QueryParameterNames.ReturnUrl);
    if (fromQuery && !fromQuery.startsWith(`${window.location.origin}/`)) {
        // This is an extra check to prevent open redirects.
        throw new Error("Invalid return url. The return url needs to have the same origin as the
current page.")
    }
    return (state && state.returnUrl) || fromQuery || `${window.location.origin}/`;
}

redirectToRegister() {

this.redirectToApiAuthorizationPath(`${ApplicationPaths.IdentityRegisterPath}?${QueryParameterNames.ReturnUrl}=${encodeURIComponent(ApplicationPaths.Login)}`);
}

redirectToProfile() {
    this.redirectToApiAuthorizationPath(ApplicationPaths.IdentityManagePath);
}

redirectToApiAuthorizationPath(apiAuthorizationPath) {
    const redirectUrl = `${window.location.origin}/${apiAuthorizationPath}`;
    // It's important that we do a replace here so that when the user hits the back arrow on the
```

```
// browser they get sent back to where it was on the app instead of to an endpoint on this
// component.
window.location.replace(redirectUrl);
}

navigateToReturnUrl(returnUrl) {
  // It's important that we do a replace here so that we remove the callback uri with the
  // fragment containing the tokens from the browser history.
  window.location.replace(returnUrl);
}
}
```