МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме

Студент	_____ (подпись, дата)	Семенков Данила Алексеевич (ФИО полностью)
Руководитель	_____ (подпись, дата)	Такташкин Д.В. (фамилия, инициалы)
Нормоконтролёр	_____ (подпись, дата)	Такташкин Д.В. (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от __.____№_____)

Заведующий кафедрой	_____ (подпись)	Козлов А.Ю. (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК	_____ (подпись)	Попова Н.А. (фамилия, инициалы)

Пенза, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет Вычислительной техники

Кафедра МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«___» _____ 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Семенков Данила Алексеевич гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме

Тема утверждена приказом ПГУ №_____ от "_____"_____20_____

3. Руководитель работы доцент кафедры «МО и ПЭВМ» Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение: разработка мобильного приложения для запоминания английских слов с Применением системы Лейтнера в игровой форме.

Основные функции системы:
1. Хранение информации об английских словах и их переводах;
2. Изучение английских слов;
3. Озвучивание заданного слова;
4. Перемещение карточек между колодами;
5. Добавление новых карточек пользователем.

Технология разработки: предметно-ориентированное программирование.
Язык программирования: C#.
Среда программирования: Unity.

5. Перечень подлежащих разработке вопросов:

1. анализ предметной области и требований к системе;
2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 16.05.2023	
5	Оформление пояснительной записки	20%	17.05.2023 – 01.06.2023	

Дата выдачи задания "_____"_____20_____

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить _____
Декан факультета (подпись, дата)

Реферат

Пояснительная записка содержит 71 лист, 13 рисунков, 10 таблиц, 2 листинга, 14 использованных источников и 5 приложений.

C#, ИГРЫ, UNITY, СИСТЕМА ЛЕЙТНЕРА, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, АНГЛИЙСКИЙ ЯЗЫК, ДИАГРАММЫ UML.

Объект работы: мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме.

Цель работы: разработка мобильного приложения для запоминания английских слов с применением системы Лейтнера в игровой форме.

Технология разработки: Среда разработки Unity.

Результаты работы: спроектировано и разработано мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме.

Область применения: люди, имеющие устройства с операционной системой Android.

Содержание

Введение

В современном мире, где глобализация и связанные с ней коммуникационные возможности становятся все более важными, знание английского языка открывает широкие горизонты для личного и профессионального развития. Однако, запоминание и активное использование новых слов часто представляют собой сложную задачу для обучающихся.

Для многих людей традиционные методы изучения языка, такие как заучивание слов наизусть или чтение учебников, неэффективны и скучны. В этой ситуации мобильные приложения, сочетающие технологии и игровые механики, становятся отличным способом сделать процесс изучения английского языка интересным, увлекательным и эффективным.

Одним из таких инновационных подходов является использование системы Лейтнера в мобильных приложениях для запоминания английских слов. Система Лейтнера основана на принципе повторения с интервалами времени, который помогает закреплять информацию в памяти более эффективно. Приложение на основе этой системы предлагает пользователю игровую среду, где новые слова представлены в интересной и запоминающейся форме, а повторения происходят в оптимальные моменты, чтобы максимизировать усвоение материала.

Игровой подход к изучению английских слов через мобильное приложение на основе системы Лейтнера предлагает ряд преимуществ. Во-первых, игра стимулирует мозговую активность и повышает мотивацию, поскольку ученики получают удовольствие от достижения новых результатов и преодоления сложностей. Во-вторых, мобильность и доступность приложения позволяют учиться в любое время и в любом месте, с помощью смартфона или планшета.

В результате выполнения работы была разработано мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме.

1 Постановка задачи и анализ аналогов разрабатываемого приложения

1.1 Анализ предметной области и постановка задачи

Система Лейтнера (англ. Leitner system) – широко используемый метод для эффективного запоминания и повторения с помощью флэш-карточек, предложенный немецким ученым и журналистом Себастьяном Лейтнером в 70-е годы XX века [1].

Основой системы Лейтнера являются так называемые флэш-карточки, на которых записывается информация для запоминания. Карточки могут быть обычными бумажными либо электронными.

Принцип использования бумажных и электронных карточек идентичен. Разница лишь в том, каким способом вы будете «тасовать колоду»: перелистывая экран или перекладывая карточки из одной стопки в другую.

Система Лейтнера позволяет не тратить время на повторение уже выученного материала, даже если он идет в связке с пока неизученным. Эта система удобна тем, что можно сосредоточиться только на том, что пока никак не запоминается, и регулировать частотность повторений по мере усвоения.

Алгоритм использования флэш-карточек:

– распределяем весь изучаемый в данный момент материал по карточкам в режиме 1 карточка = 1 единица информации;

– карточки с информацией, которую не получается запомнить совсем, складываем в первую колоду и повторяем ежедневно;

– карточки с информацией, которую вы помните фрагментарно, помещаете во вторую колоду и повторяете через день;

– карточки с информацией, которую вы иногда забываете или не всегда можете быстро вспомнить, помещаете в третью колоду и повторяете раз в три дня;

– по мере освоения материала карточки из первой колоды перекладываете во вторую, а затем в третью колоду [2].

Мобильное приложение должно иметь функцию хранения информации об английских словах и их переводах: Приложение позволяет хранить и организовывать информацию об английских словах и их переводах. Также пользователь может добавлять новые слова, указывать их переводы и иконки к этим словам. Вся эта информация будет сохранена и доступна для изучения и использования.

Помимо этого, необходимо реализовать перемещение карточек между колодами: Приложение подвластно системе Лейтнера и предлагает функцию перемещения карточек между различными колодами. Карточки содержат английские слова и их переводы, и пользователь сможет организовывать их в группы в соответствии со своими предпочтениями. Это поможет ему структурировать изучение слов и сделать процесс более эффективным.

1.2 Анализ программ аналогов

Перед началом разработки системы необходимо провести поиск и анализ уже существующих аналогов, чтобы учесть их сильные и слабые стороны, а также определить потребности пользователей, которые могут быть ещё не учтены в таком типе продукта.

Brainscape – сервис для создания и обучения всему на свете с помощью флеш-карточек. Вы можете создавать свои уроки или присоединиться к уже созданным учебным программам [3].

В приложении можно найти карточки практически по любой тематике, но несмотря на такое большое разнообразие большинство тем на английском языке и есть платные карточки.

Quizlet – это сервис, который позволяет легко запоминать любую информацию, которую можно представить в виде учебных карточек. Все что требуется – это найти в базе или создать интерактивный материал – собственные карточки, добавляя к ним картинки и аудиофайлы и затем выполнять упражнения, чтобы запомнить данный материал [4].

В приложении можно найти карточки на разные темы, но на самих карточках нет картинок, только текст, а также есть платные темы.

Карточки для запоминания. Карточки для запоминания – это приложение для эффективного запоминания слов на основе интервальных повторений. Приложение имеет простой интерфейс, но в нём нет никаких готов тем и карточек.

Рассмотрев схожие по функционалу программы, были выделены критерии для их сравнения:

– доступность – показатель наличия платного контента и его количества в приложении;

– разнообразие интерфейса – показатель разнообразия интерфейса, наличия картинок;

– простота интерфейса – показатель простоты интерфейса;

– готовые темы со словами – показатель, отвечающий за наличие готового материала для изучения.

Для оценки критериев используются цифры от 1 до 3, где 1 – низкое проявление критерия, а 3 – высокое. Результат оценки приведен в таблице 1.

Таблица 1 – Сравнение программ аналогов

	Brainscape	Quizlet	Карточки для запоминания
доступность	1	1	1
разнообразие интерфейса	1	1	1
простота интерфейса	1	1	2
готовые темы со словами	3	3	1

По результатам сравнения программ аналогов будут определены функциональные требования для разрабатываемой системы. Также было выявлено отсутствие индивидуальность карточек за счёт того, что у них нет картинок. Поэтому разработка бесплатного приложения с уникальными карточками за счёт картинок будет более доступна для пользователей.

1.3 Анализ функциональных требований

На основе поставленной задачи и сравнения аналогов были определены основные требования к разрабатываемому приложению. Мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме должно иметь следующие основные функции:

– хранения информации об английских словах и их переводах;
– изучения английских слов;
– озвучивания заданного слова;
– перемещения карточек между колодами;
– добавления новых карточек пользователем.

Для визуализации всех основных функций приложения была разработана диаграмма вариантов использования, представленная в приложении А.

Диаграмма вариантов использования (use case diagram) – диаграмма, на которой изображаются отношения между актерами и вариантами использования [5].

Диаграмма вариантов использования нужна для описания взаимодействия между актерами (пользователями) и системой, идентификации функциональных требований, проверки сценариев использования, улучшения пользовательского опыта и согласования изменений в системе.

Далее описаны элементы, изображенные на диаграмме вариантов использования.

Актером является пользователь приложения.

Вариант использования «Добавить новое слово» заключается в добавлении нового слова в хранилище слов приложения. Для того чтобы добавить новое слово, необходимо ввести слово на английском, его перевод, а также указать картинку, соответствующую этому слову.

Вариант использования «Изучать слова» заключается в изучение и последующем добавлении пользователем слов в колоды для интервальных повторений. Для этого пользователь выбирает темы для изучения, нажимает

кнопку начать и изучает слова в игровой форме, посредством выбора правильного перевода слова из списка слов. Далее после выбора ответа, пользователь определяет в какую колоду попадёт данное слово.

Вариант использования «Изучать слова» заключается в повторении слов с помощью системы Лейтнера. Пользователь выбирает колоду для повторения, затем посредством выбора правильного перевода слова из списка слов, он определяет останется ли это слово в текущей колоде, либо переместиться в другую. Также пользователь имеет возможность озвучить повторяемое слово для его лучшего запоминания.

1.4 Обоснование выбора инструментальных средств проектирования и разработки

Для проектирования UML - диаграмм мобильного приложения будет использоваться такое средство как StarUML, которое позволяет строить множество UML диаграмм.

StarUML – программный инструмент моделирования, который поддерживает UML (Унифицированный язык моделирования). Программное средство поддерживает одиннадцать различных типов диаграмм, принятых в нотации UML 2.0. Среда разработки StarUML превосходно настраивается в соответствии с требованиями пользователя и имеет высокую степень расширяемости, особенно в области своих функциональных возможностей [6].

Для разработки мобильного приложения для запоминания английских слов с применением системы Лейтнера в игровой форме будет использоваться кроссплатформенная среда разработки Unity.

Unity – это игровой движок, на котором разрабатывают мобильные игры и проекты для ПК (Windows, iOS, Linux) и консолей, например для Xbox, PlayStation. В нем есть разные компоненты для работы с графикой, анимацией, физикой объектов, звуком, шаблонами и скриптами.

Это удобный бесплатный инструмент для начинающих разработчиков, в нем можно создавать проекты в одиночку [7].

2 Планирование и расчет бюджета разрабатываемого приложения

Оценка бюджета разработки мобильного приложения является важным этапом при планировании проекта. Она помогает определить финансовые ресурсы, необходимые для создания и запуска приложения, и обеспечивает понимание общей стоимости разработки.

При оценке бюджета разработки мобильного приложения следует учитывать несколько ключевых факторов.

Функциональность приложения: чем больше функциональных возможностей и сложных компонентов требуется в приложении, тем больше затрат на разработку. Необходимо провести детальный анализ требований и определить, какие функции приложения являются наиболее важными и необходимыми для первоначального выпуска, а какие могут быть добавлены позже в виде обновлений.

Дизайн и пользовательский интерфейс: Привлекательный и интуитивно понятный дизайн является важным фактором успеха мобильного приложения. Однако профессиональное создание дизайна и разработка пользовательского интерфейса требуют дополнительных затрат на дизайнеров и UX/UI-специалистов.

Платформы и устройства: если планируется выпуск приложения для нескольких платформ (например, iOS и Android), то необходимо учесть затраты на разработку и адаптацию приложения под каждую из платформ. Кроме того, существуют различные устройства с разными размерами экранов, и их поддержка может потребовать дополнительных усилий.

Тестирование и обеспечение качества: критически важно провести тщательное тестирование приложения, чтобы обнаружить и исправить ошибки и неполадки. Поддержка и обновление приложения также могут потребовать финансовых ресурсов.

Масштабирование и поддержка: если планируется дальнейшее развитие и масштабирование приложения после его запуска, следует учесть

дополнительные расходы на поддержку, обновления и улучшения функциональности.

Для планирования процесса разработки и расчета бюджета можно использовать диаграмму Ганта.

Диаграмма Ганта – это визуальное представление графика работ, построенное согласно плану проекта. На ней отражены задачи и последовательность их выполнения.

График работ состоит из ряда отрезков, размещённых вдоль временной оси. Каждый из них соответствует отдельной задаче или подзадаче. Начало и конец отрезка соответствуют моменту начала и завершения работы по задаче. Длина отрезка – продолжительность работ [8].

Для построения диаграммы необходимо выделить задачи, а также сроки их выполнения. Задачи проекта представлены в таблице 2.

Таблица 2 – Задачи проекта

Название задачи	Длительность задачи в днях	Дата начала	Дата окончания
Начало проекта	0	12.04.2023	12.04.2023
Постановка задачи и анализ аналогов разрабатываемого приложения	4	12.04.2023	16.04.2023
Анализ программ аналогов	2	12.04.2023	14.04.2023
Анализ функциональных требований	1	14.04.2023	15.04.2023
Выбор инструментальных средств проектирования и разработки	1	15.04.2023	16.04.2023
Проектирование мобильного приложения	9	17.04.2023	26.04.2023
Проектирование архитектуры приложения	4	17.04.2023	21.04.2023
Проектирование пользовательского интерфейса	5	21.04.2023	26.04.2023
Реализация приложения	14	27.04.2023	11.05.2023
Кодирование	11	27.04.2023	08.05.2023
Реализация пользовательского интерфейса	3	08.05.2023	11.05.2023
Тестирование приложения	4	12.05.2023	16.05.2023
Тестирование методом черного ящика	1	12.05.2023	13.05.2023
Тестирование метрикой Холстеда	1	13.05.2023	14.05.2023
Автоматизированное тестирование	2	14.05.2023	16.05.2023

Продолжение таблицы 2

Название задачи	Длительность задачи в днях	Дата начала	Дата окончания
Создание пояснительной записки	16	17.05.2023	01.06.2023
Написание пояснительной записки	9	17.05.2023	26.05.2023
Оформление и печать пояснительной записки	7	26.05.2023	01.06.2023
Конец проекта	0	01.06.2023	01.06.2023

На основе выделенных задач и сроков была построена диаграмма Ганта, представленная на рисунке 1.

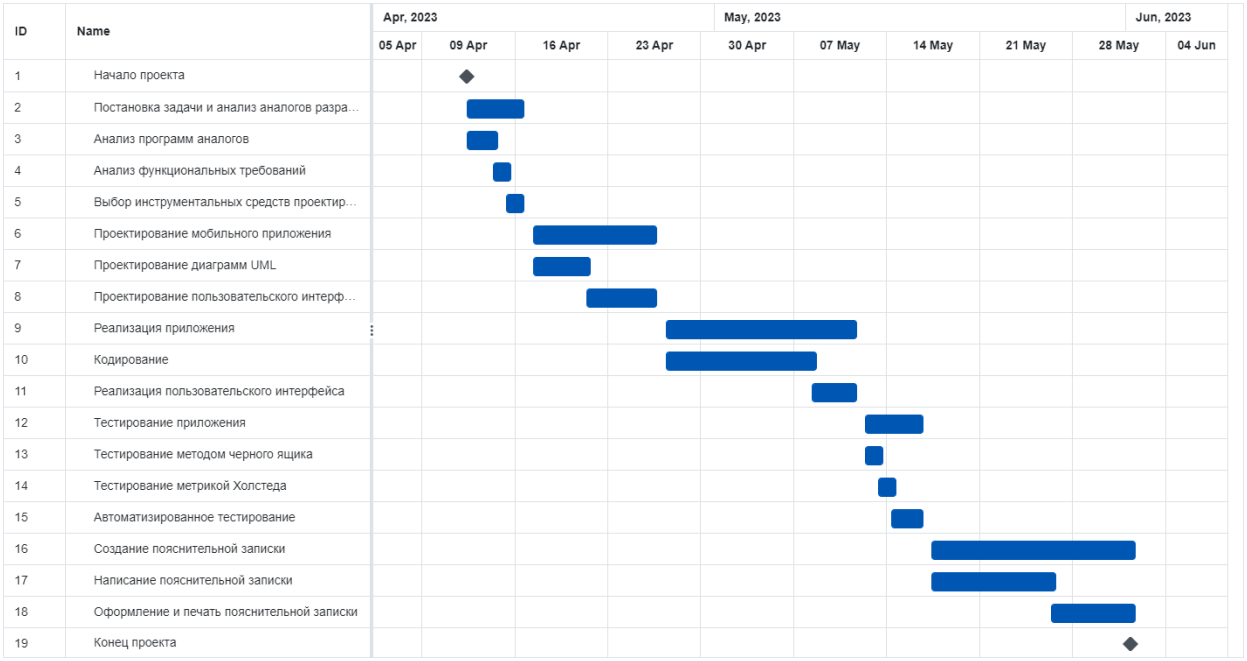


Рисунок 1 – Диаграмма Ганта

Для подсчета затрат на разработку необходимо определить ресурсы проекта. Выделенные ресурсы приведены в таблице 3.

Таблица 3 – Ресурсы проекта

Название ресурса	Тип	Стандартная ставка	Начисление
Аналитик	Трудовой	100,00 ₽/ч	Пропорциональное
Проектировщик	Трудовой	200,00 ₽/ч	Пропорциональное
Программист	Трудовой	200,00 ₽/ч	Пропорциональное
Дизайнер	Трудовой	150 ₽/ч	Пропорциональное
Тестировщик	Трудовой	100 ₽/ч	Пропорциональное
Компьютер	Материальный	50 000,00 ₽	В начале
Смартфон	Материальный	15 000,00 ₽	В начале
Бумага	Материальный	500,00 ₽	В начале
Принтер	Материальный	5 000,00 ₽	В начале

Для описанных ранее задач, на основе ресурсов, была рассчитана стоимость, необходимая для выполнения каждого этапа разработки. Результат представлен в таблице 4.

Таблица 4 – Задачи и затраты проекта

Название задачи	Длительность задачи в днях	Название ресурса	Затраты
Начало проекта	0	Компьютер, Смартфон, Бумага, Принтер	70 500,00 ₽
Постановка задачи и анализ аналогов разрабатываемого приложения	4		**3 200,00 ₽**
Анализ программ аналогов	2	Аналитик	1 600,00 ₽
Анализ функциональных требований	1	Аналитик	800,00 ₽
Выбор инструментальных средств проектирования и разработки	1	Аналитик	800,00 ₽
Проектирование мобильного приложения	9		**20 400,00 ₽**
Проектирование архитектуры приложения	4	Проектировщик	6 400,00 ₽
Проектирование пользовательского интерфейса	5	Проектировщик, Дизайнер	14 000,00 ₽
Реализация приложения	14		**26 000,00 ₽**
Кодирование	11	Программист	17 600,00 ₽
Реализация пользовательского интерфейса	3	Программист, Дизайнер	8 400,00 ₽
Тестирование конфигурации	4		**3 200,00 ₽**
Тестирование методом черного ящика	1	Тестировщик	800,00 ₽
Тестирование метрикой Холстеда	1	Тестировщик	800,00 ₽
Автоматизированное тестирование	2	Тестировщик	1 600,00 ₽
Создание пояснительной записки	16		**12 800,00 ₽**
Написание пояснительной записки	9	Аналитик	7 200,00 ₽
Оформление и печать пояснительной записки	7	Аналитик	5 600,00 ₽
Конец проекта	0		

Таким образом, по результатам оценки бюджета разработки мобильного приложения была составлена диаграмма Ганта, рассчитано время необходимое на полный цикл разработки проекта, которое составляет 47 дней, на основании необходимых ресурсов рассчитан бюджет проекта, который равен 136 100 рублей.

3 Проектирование мобильного приложения

Проектирование мобильного приложения является важным этапом разработки, который определяет его функциональность, интерфейс и пользовательский опыт. Вот несколько ключевых аспектов, которые следует учитывать при проектировании мобильного приложения.

Определение целей и функциональности: необходимо определить основные цели приложения и функции, которые оно должно выполнять. Составить список основных функций и определить, как они будут интегрированы в пользовательский интерфейс.

Информационная архитектура: необходимо разработать структуру приложения и определить, как пользователь будет перемещаться по его различным разделам и функциям. Обеспечить логическую и интуитивно понятную навигацию, чтобы пользователи могли легко находить нужную информацию и выполнять задачи.

Дизайн интерфейса: необходимо разработать привлекательный и интуитивно понятный пользовательский интерфейс (UI), который соответствует бренду приложения и предоставляет хороший пользовательский опыт (UX). Обратить внимание на компоненты дизайна, такие как цветовая гамма, шрифты, иконки и расположение элементов интерфейса.

Адаптивность и оптимизация: необходимо учитывать различные разрешения экранов и устройств, на которых будет работать приложение. Убедится, что интерфейс адаптируется к разным размерам экранов и оптимизирован для быстрой загрузки и производительности.

Важно помнить, что проектирование мобильного приложения – это искусство, которое требует внимания к деталям, понимания потребностей пользователей и постоянного улучшения на основе обратной связи.

В результате проектирования были выбраны используемые подходы, продумана архитектура мобильного приложения и проработан графический интерфейс.

3.1 Проектирование структуры мобильного приложения

Структура разрабатываемого приложения была спроектирована с использованием шаблона проектирования Dependency Injection Pattern.

Паттерн внедрение зависимостей (Dependency Injection Pattern) – это шаблон проектирования, который применяет принцип IoC, чтобы гарантировать, что класс абсолютно никаким образом не участвует в создании экземпляра зависимого класса, и тем более не управляет его жизненным циклом (lifetime), то есть временем существования объектов. Другими словами, забота о создании класса и наполнение переменных его экземпляра, который «пришел» через конструктор, или через метод – целиком и полностью лежит на платформе. То есть, где-то на более высоком уровне. Говоря про Dependency Injection Pattern нельзя не сказать про Inversion of Control, который был упомянут выше.

Инверсия управления (Inversion of Control, IoC) – важный принцип объектно-ориентированного программирования, используемый для уменьшения зацепления (связанности) в компьютерных программах. Также архитектурное решение интеграции, упрощающее расширение возможностей системы, при котором поток управления программы контролируется фреймворком.

Внедрение зависимости используется во многих фреймворках, которые называются IoC-контейнерами.

Dependency Injection (DI) Contaier – это инструмент, который может решать (resolve) зависимости для их внедрения. Говоря простыми словами, это «черный ящик», в котором можно зарегистрировать классы (интерфейсы и их реализации) для дальнейшего их решения (resolve) в нужных местах, например в конструкторах. Кстати, надо сказать, что внедрение зависимостей возможно не только через конструктор, но и через методы и свойства. Хотя внедрение через конструктор самое распространённое внедрение [9].

Структура приложения была разделена на следующие части: сервисы, репозитории, контроллеры и дескрипторы.

Рассмотрим наиболее важную часть разрабатываемого приложения – сервисы.

Для визуализации всех сервисов приложения была разработана диаграмма классов, представленная в приложении Б.

Диаграмма классов в языке моделирования UML относится к структурному типу диаграмм, используется для визуализации структуры классов в системе, атрибутов, методов, интерфейсов и отношений между ними. Применяется при проектировании архитектуры, документировании системы, уточнении требований, а также для поддержки системы.

Диаграмма классов представляет описание структуры классов в системе и их взаимосвязи. Она отображает как статические аспекты системы, включая классы, атрибуты и методы, а также динамические аспекты, такие как связи между объектами и выполнение методов во время выполнения программы [10].

Рассмотрим часть структуры приложения (приложение Б), отвечающей за реализацию сервисов. Данная часть включает в себя следующие классы:

– класс «DependencyService», назначение – реализует сервис для работы с Dependency Injection Contaier;

– класс «DescriptorService», назначение – реализует сервис для работы со словами и их переводами, зарегистрированными в системе;

– класс «MainMenuService», назначение – реализует сервис для работы с пользовательским интерфейсом;

– класс «ExploreService», назначение – реализует сервис для работы с системой Лейтнера;

– класс «SaveWordsService», назначение – реализует сервис для работы с сохраненем слов, добавленных пользователем;

– класс «AudioService», назначение – реализует сервис для работы с озвучиванием английских слов.

Рассмотрим подробнее спецификацию классов.

Методы класса «DependencyService»:

– «AddControllerToObject», типы данных аргументов – «GameObject», тип возвращаемого значения – «MonoBehaviour»;

– «CreateObjectWithController», типы данных аргументов – «GameObject», string, тип возвращаемого значения – «MonoBehaviour»;

– «BuildObject», типы данных аргументов – «GameObject», тип возвращаемого значения – «MonoBehaviour»;

Метод «BuildObject» используется для обращения к Dependency Injection Contaier, который собирает класс со всеми его зависимостями.

Метод «AddControllerToObject» используется для обращения к методу «BuildObject», для того чтобы добавить класс контроллер к игровому объекту.

Метод «CreateObjectWithController» используется для создания нового объекта и обращения к методу «BuildObject», для того чтобы добавить класс контроллер к игровому объекту.

Методы класса «DescriptorService»:

– метод «LoadDescriptors» используется для загрузки английских слов и их переводах из xml файла;

– метод «GetAllDescriptors» используется для получения всех дескрипторов, классов, в которых содержится информация об английских словах и их переводах;

– метод «GetDescriptorByWords» используется для получения всех дескрипторов, английские слова которых содержатся в переданном параметре;

– метод «GetDescriptorsWithWordType» используется для получения всех дескрипторов, у которых тип слова содержатся в переданном параметре.

Методы класса «MainMenuService»:

– метод «CreateMainMenu» используется для создания объекта главного меню вместе с освоим контроллером.

Методы класса «ExploreService»:

– метод «StartExploreAsync» используется для изучения и повторения слов по системе Лейтнера;

– метод «CreateCardControllers» используется для создания флэш-карточек, на которых находится информация о заданном слове;

– метод «PrepareCardController» используется для подготовки контроллера флэш-карточки перед её показом;

– метод «ShowDecks» используется для отправки события ui компонентам о том, что необходимо показать колоды;

– метод HideDecks используется для отправки события ui компонентам о том, что необходимо скрыть колоды;

Методы класса «SaveWordsService»:

– метод «SaveWord» используется для сохранения слова, добавленного пользователем.

Методы класса «AudioService»:

– метод «CreateAudioSourceController» используется для объекта «AudioSourceController», который проигрывает звук.

– метод «GetAudio» используется для получения звукового файла выбранного слова.

Все сервисы зарегистрированы в Dependency Injection Contaier, с помощью которого могут быть доступны в любом компоненте программы.

3.2 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса (UI) является важной составляющей разработки любого продукта, будь то веб-сайт, мобильное приложение или программное обеспечение. Вот некоторые ключевые аспекты, которые следует учитывать при проектировании пользовательского интерфейса.

Исследование пользователей: перед началом проектирования необходимо провести исследование пользователей и понять их потребности, характеристики и поведение. Это поможет создать интерфейс, который будет удобным и интуитивно понятным для целевой аудитории.

Иерархия и структура: необходимо разработать ясную иерархию информации и структуру интерфейса. Определить основные разделы, категории и функции, и организовать их в логическом порядке. Это поможет пользователям быстро ориентироваться и находить нужную информацию.

Навигация: необходимо создать интуитивно понятную навигацию, чтобы пользователи могли легко перемещаться по интерфейсу. Использовать понятные метки, меню, иконки или жесты, чтобы обеспечить плавное и эффективное взаимодействие с приложением.

Визуальный дизайн: необходимо разработать привлекательный и согласованный визуальный дизайн, соответствующий бренду и целям продукта. Учитывая такие факторы, как цветовая гамма, шрифты, иконки и компоновка элементов.

В данной работе было проведено проектирование графического интерфейса мобильного приложения для запоминания английских слов с применением системы Лейтнера в игровой форме. Главная цель проектирования – создание удобного и интуитивно понятного интерфейса, который позволит пользователям легко и быстро пользоваться приложением.

В результате проектирования были созданы следующие макеты экранов.

На рисунке 2 изображён макет начального экрана.

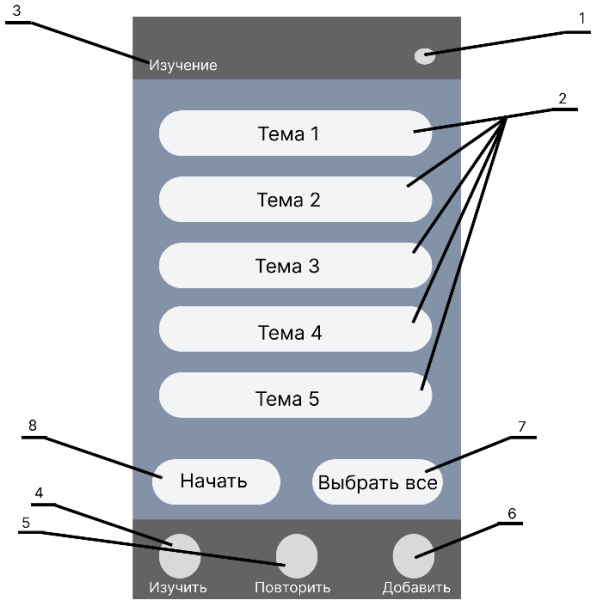


Рисунок 2 – Макет начального экрана

Макет стартового экрана содержит следующие элементы:

1 – кнопка подсказки;

2 – toggle кнопки с выбором тем для изучения слов;

3 – заголовок экрана;

4 – кнопка «Изучить» при нажатии на которую откроется экран изучения новых слов;

5 – кнопка «Повторить» при нажатии на которую откроется экран повторения изученных слов;

6 – кнопка «Добавить» при нажатии на которую откроется экран добавления новых слов;

7 – кнопка «Выбрать все» при нажатии на которую выбираются все темы для изучения слов;

8 – кнопка «Начать» при нажатии на которую начинается изучение новых слов.

На рисунке 3 изображён макет экрана повторения слов.

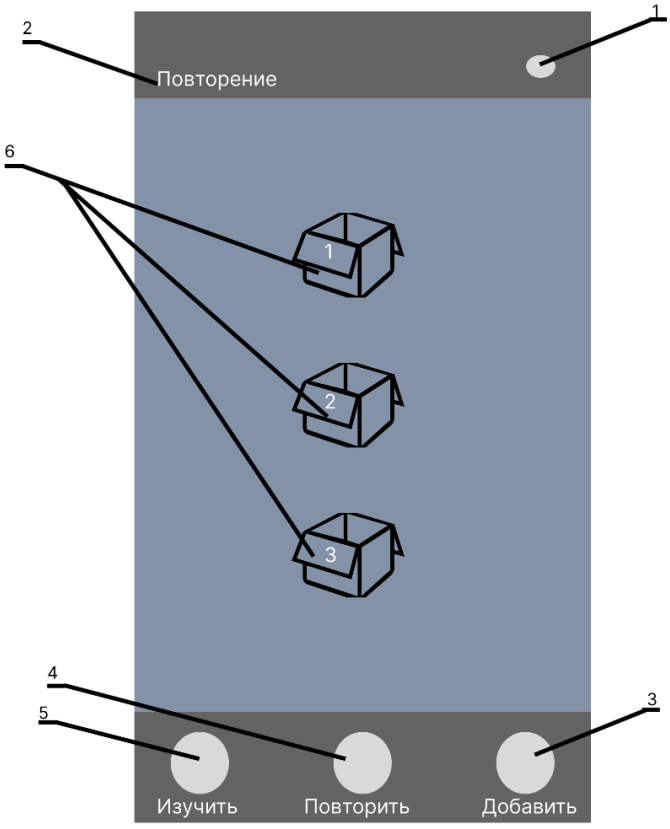


Рисунок 3 – Макет экрана повторения слов

Макет экрана повторения слов содержит следующие элементы:

1 – кнопка подсказки;

2 – заголовок экрана;

3 – кнопка «Добавить» при нажатии на которую откроется экран добавления новых слов;

4 – кнопка «Повторить» при нажатии на которую откроется экран повторения изученных слов;

5 – кнопка «Изучить» при нажатии на которую откроется экран изучения новых слов;

6 – кнопки колод, подчиняющихся системе Лейтнера.

На рисунке 4 изображён макет экрана добавления новых слов.

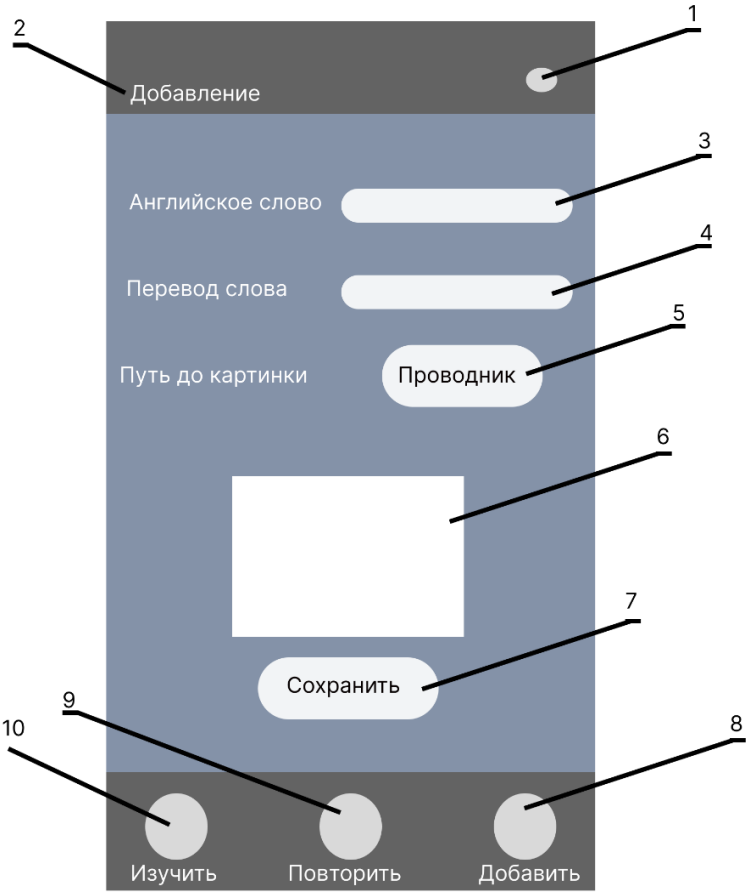


Рисунок 4 – Макет экрана добавления новых слов

Макет экрана добавления новых слов содержит следующие элементы:

1 – кнопка подсказки;

2 – заголовок экрана;

3 – поле ввода английского слова;

4 – поле ввода перевода английского слова;

5 – кнопка «Проводник», при нажатии на которую открывается проводник для выбора изображения;

6 – изображение, в котором будет отображаться картинка, которую пользователь выбрал в проводнике;

7 – кнопка «Сохранить», при нажатии на которую будет сохраняться новое слово;

8 – кнопка «Добавить» при нажатии на которую откроется экран добавления новых слов;

9 – кнопка «Повторить» при нажатии на которую откроется экран повторения изученных слов;

10 – кнопка «Изучить» при нажатии на которую откроется экран изучения новых слов.

На рисунке 5 изображён макет экрана выбора правильного ответа.

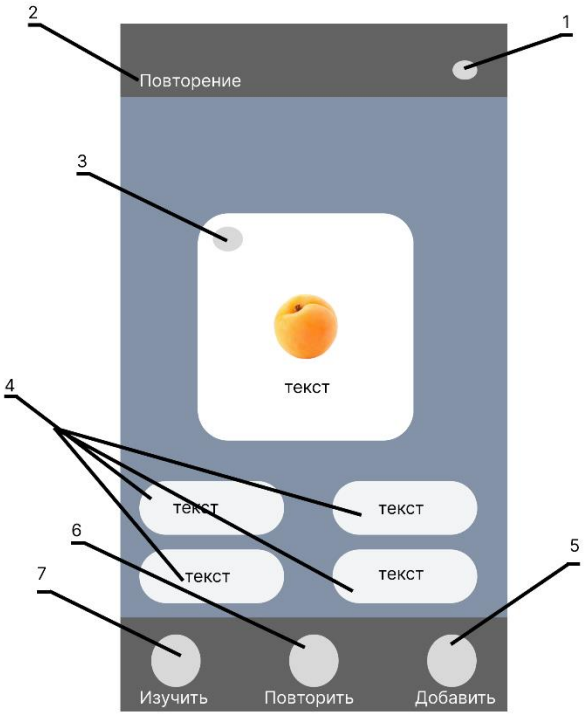


Рисунок 5 – Макет экрана выбора правильного ответа

Макет экрана выбора правильного ответа содержит следующие элементы:

1 – кнопка подсказки;

2 – заголовок экрана;

3 – кнопка озвучивания слова;

4 – кнопки вариантов ответов;

5 – кнопка «Добавить» при нажатии на которую откроется экран добавления новых слов;

6 – кнопка «Повторить» при нажатии на которую откроется экран повторения изученных слов;

7 – кнопка «Изучить» при нажатии на которую откроется экран изучения новых слов.

Таким образом, были спроектированы все экраны приложения. В дальнейшем на основе этих макетов будет разработан пользовательский интерфейс.

4 Реализация мобильного приложения

4.1 Реализация кода мобильного приложения

Код приложения был написан на языке C#.

Компоненты приложения приведены на диаграмме компонентов в приложении В.

Диаграмма компонентов предназначена для описания физического представления системы. На диаграмме устанавливаются зависимости между компонентами, тем самым определяя архитектуру приложения. Компонентом является физически существующая часть системы, например коды модулей или командные файлы. Компоненты связаны друг с другом при помощи отношения зависимости, то есть изменения одного объекта влияет на поведение другого [11].

Подробнее рассмотрим каждый из используемых компонентов (приложение В).

Компоненты «DecksRepository», «TimeDeckRepository», «SaveNewWordR eposity» необходимы для хранения локальных данных пользователя.

Компонент «DependencyService» необходим для работы с Dependency Injection Contaier.

Компонент «DescriptorService» необходим для работы со словами и их переводами, зарегистрированными в приложении.

Компонент «MainMenuService» необходим для работы с пользовательским интерфейсом.

Компонент «ExploreService» необходим для работы с системой Лейтнера.

Компонент «SaveWordsService» необходим для работы с сохранением слов, добавленных пользователем.

Компонент «AudioService» необходим для работы с озвучиванием английских слов.

Исходный код компонентов приложения приведен в приложении Г.

4.2 Реализация пользовательского интерфейса

В соответствии с требованиями к интерфейсу, представленными в разделе проектирования, был разработан пользовательский интерфейс, необходимый для взаимодействия с приложением.

Рассмотрим основные экраны приложения:

– начального экран;

– экран повторения слов;

– экран добавления новых слов;

– экран выбора правильного ответа.

На начальном экране приложения (рисунок 6) по умолчанию отображается экран изучения новых слов, а также кнопки перехода по другим экранам.

Рисунок 6 – Начальный экран

При нажатии на кнопку «Повторить» отображается экран повторения слов, в котором находятся 3 колоды для повторения слов по системе Лейтнера (рисунок 7).

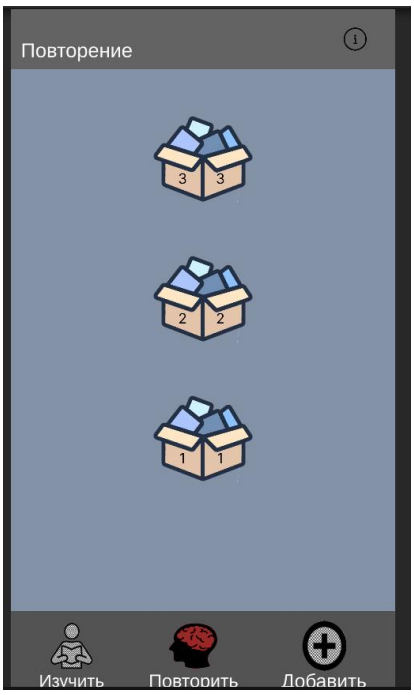


Рисунок 7 – Экран повторения слов

При нажатии на кнопку «Добавить» отображается экран добавления новых слов (рисунок 8).

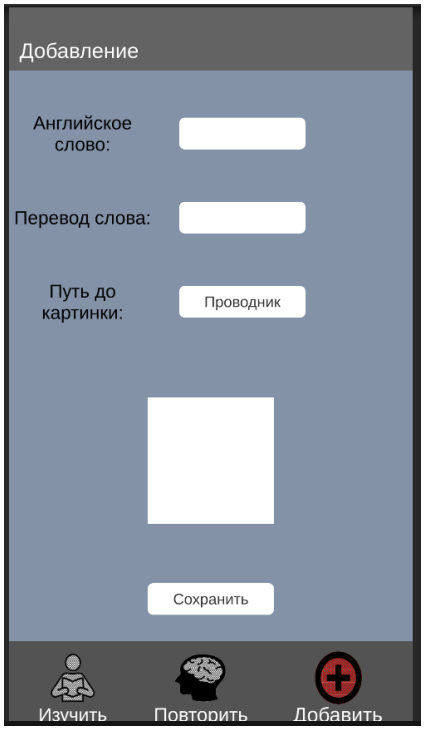


Рисунок 8 – Экран добавления новых слов

При выборе колоды для повторения на экране повторения слов будет отображаться экран выбора правильного ответа (рисунок 9).

Рисунок 9 – Экран выбора правильного ответа

Таким образом в соответствии с требованиями к интерфейсу, представленными в разделе проектирования был разработан пользовательский интерфейс, необходимый для взаимодействия с приложением.

5 Тестирование документов конфигурации

В ходе анализа способов тестирования программного обеспечения было определено, что более подходящими из них для данного приложения являются: тестирование методом черного ящика, тестирование метрики кода, а также автоматизированное тестирование.

5.1 Тестирование формы добавления новых слов

Black – box тестирование – это функциональное и нефункциональное тестирование без доступа к внутренней структуре компонентов системы. Метод тестирования «черного ящика» – процедура получения и выбора тестовых случаев на основе анализа спецификации (функциональной или нефункциональной), компонентов или системы без ссылки на их внутреннее устройство [12].

Для реализации тестирования методом черного ящика были использованы функциональная диаграмма, классы эквивалентности и таблица решений.

Классом эквивалентности называется набор данных, который запускает одни и те же модули и должен приводить к одним и тем же результатам.

Любые данные в рамках класса эквивалентны, это означает что если один тест-кейс в кассе эквивалентности обнаружил/не обнаружил дефект, то все остальные тест-кейсы внутри этого класса эквивалентности обнаружат/не обнаружат тот же самый дефект [13].

В приведенном примере тестируется форма для ввода новых слов, которая предусматривает, что введенное слово должно быть длиной от 3 до 20 символов включительно, должно начинаться с буквы и состоять из букв латинского алфавита или кириллицы.

Для того, чтобы в дальнейшем была возможность составить тест-кейс, требуется обозначить все классы эквивалентности, которые делятся на правильные и неправильные.

Представление классов эквивалентности для тестирования формы ввода новых слов представлено в таблице 5.

Таблица 5 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина названия	3-20 (1)	0-3 (2) 21 - ∞ (3)
Структура	Содержатся только буквы (4)	Содержатся не только буквы (5)
Содержание	Буквы Кириллица (6) Буквы Латиница (7) Пробел (8)	Остальные (9)

Графическое представление для классов эквивалентности представлено на рисунке 10.

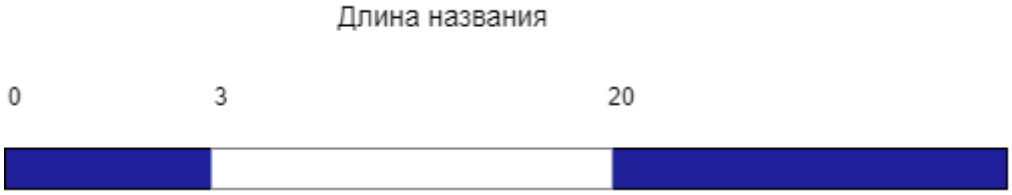


Рисунок 10 – Графическое представление

Структура для классов эквивалентности представлена на рисунке 11.

Рисунок 11 – Структура для классов эквивалентности

Для классов эквивалентности были составлены тест-кейсы, которые представлены в таблице 6.

Таблица 6 – Тест кейсы

№	КЭ	Входные данные	Выходные данные
1	Все правильные (1, 4, 6, 7, 8)	adaptative	Успешно
2	2, 4, 6, 7	da	Ошибка: меньше 3х символов
3	3, 4, 6, 7	ddfghjklqwertyuiopdgg	Ошибка: больше 20 символов
4	5, 7, 8	dagag2	Ошибка: Содержатся не только буквы
5	1, 4, 9	Schoo][l	Ошибка: Некорректные символы.

Таблица причин и следствий, построенная для тестирования формы добавления нового слова, представлена в таблице 7.

Таблица 7 – Таблица причин и следствий

Причины	
П1	Слово на английском введено
П2	Введённое слово на английском корректно
П3	Перевод слова на русском введён
П4	Введённое слово на русском корректно
П5	Картинка для слова добавлена
Следствия	
С1	Сообщение: «Введите слово на английском»
С2	Сообщение: «Введенное слово некорректно»
С3	Сообщение: «Введите перевод слова на русском»
С4	Сообщение: «Введенное слово некорректно»
С5	Сообщение: «Картинка для слова не была добавлена»
С6	Сообщение: «Слово успешно добавлено»

Для успешного добавления нового слова нужно заполнить поля английское слово, перевод и картинка. Визуальное представление причин и следствий изображено на функциональной диаграмме на рисунке 12.

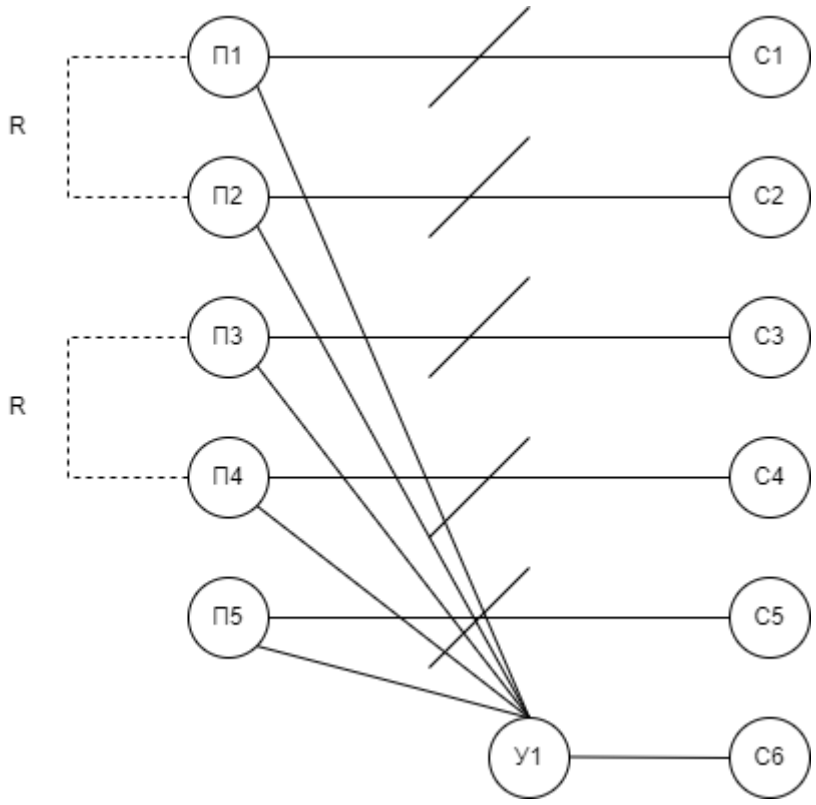


Рисунок 12 – Функциональная диаграмма причин и следствий

Все поля данной формы по добавлению нового слова являются обязательными для заполнения. Важно учитывать, что вторая причина не может быть выполнена без выполнения первой. И четвертая причина не может быть выполнена без выполнения третьей, так как перевод слова не может быть корректным, если он не введен вовсе.

В случае, если все причины успешно выполнены, то выполняется шестое следствие, которое сообщает о том, что новое слово добавлено.

На основе таблицы и диаграммы была построена таблица решений (таблица 8), позволяющая составить тесты по всем возможным комбинациям входных данных.

Таблица 8 – Таблица решений

	1	2	3	4	5	6	7	8	9
П1	0	1	1	1	1	1	1	0	0
П2	0	0	1	1	1	1	1	0	0
П3	0	0	0	1	1	1	0	1	1
П4	0	0	0	0	1	1	0	1	1

Продолжение таблицы 8

	1	2	3	4	5	6	7	8	9
П5	0	0	0	0	0	1	0	0	1
С1	1	0	0	0	0	0	0	1	1
С2	1	1	0	0	0	0	0	1	1
С3	1	1	1	0	0	0	1	0	0
С4	1	1	1	1	0	0	1	0	0
С5	1	1	1	1	1	0	1	1	0
С6	0	0	0	0	0	1	0	0	0

По полученным данным были написаны тест-кейсы, позволяющие проверить каждую комбинацию причин. Тест-кейсы представлены в таблице 9.

Таблица 9 – Тест-кейсы таблицы решений

№	Входные данные	Выходные данные
1	2	3
1	Слово на английском: Перевод: Картинка:	Сообщение: «Введите слово на английском» Сообщение: «Введенное слово некорректно» Сообщение: «Введите перевод слова на русском» Сообщение: «Введенное слово некорректно» Сообщение: «Картинка для слова не была добавлена»
2	Слово на английском: cubic1 Перевод: Картинка:	Сообщение: «Введенное слово некорректно» Сообщение: «Введите перевод слова на русском» Сообщение: «Введенное слово некорректно» Сообщение: «Картинка для слова не была добавлена»
3	Слово на английском: cubic Перевод: Картинка:	Сообщение: «Введите перевод слова на русском» Сообщение: «Введенное слово некорректно»
4	Слово на английском: cubic Перевод: кубический23 Картинка:	Сообщение: «Введенное слово некорректно» Сообщение: «Картинка для слова не была добавлена»
5	Слово на английском: cubic Перевод: кубический Картинка:	Сообщение: «Картинка для слова не была добавлена»
6	Слово на английском: cubic Перевод: кубический Картинка: куб.png	Сообщение: «Слово успешно добавлено»
7	Слово на английском: cubic Перевод: Картинка:	Сообщение: «Введите перевод слова на русском» Сообщение: «Введенное слово некорректно»

Продолжение таблицы 9

1	2	3
8	Слово на английском: Перевод: кубический Картинка:	Сообщение: «Введите слово на английском» Сообщение: «Введенное слово некорректно» Сообщение: «Картинка для слова не была добавлена»
9	Слово на английском: Перевод: кубический Картинка: куб.png	Сообщение: «Введите слово на английском» Сообщение: «Введенное слово некорректно»

Форма для добавления нового слова работает корректно и при валидации правильно выдает пользователю ошибки.

5.2 Тестирование метрики кода

Для оценки сложности функции была использованы метрика Холстеда. Данная метрика позволяет рассчитать длину, словарь, а также сложность программы на основе количества общих и уникальных операторов и операндов.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

– число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

– число уникальных операндов программы (словарь операндов);

– общее число операторов в программе;

– общее число операндов в программе [14].

Для тестирования использовалась функция валидатор, через которую проходят все заполняемые поля перед проверкой. Код представлен в листинге 1.

Листинг 1 – Код функции по валидации полей при добавлении нового слова

```
private bool ValidateParameters(string englishWord, string russianWord, string
pathToImage)
{
if (string.IsNullOrEmpty(englishWord)) {
Debug.Log("Введите слово на английском");
return false;
}
if (!Regex.IsMatch(englishWord, ENGlISH_PATTERN)) {
```

```
Debug.Log("Введенное слово некорректно");
return false;
}
if (string.IsNullOrEmpty(russianWord)) {
Debug.Log("Введите перевод слова на русском»");
return false;
}
if (!Regex.IsMatch(russianWord, RUSSIAN_PATTERN)) {
Debug.Log("Введенное слово некорректно");
return false;
}
if (string.IsNullOrEmpty(pathToImage)) {
Debug.Log("Картинка для слова не была добавлена»");
return false;
}
return true;
}
```

В качестве входных данных функция принимает слово на английском его перевод на русском и путь до картинки.

Далее проверяется, введённое слово на английском, и, если оно не корректно или его нет возвращается false и выводится соответствующее сообщение.

Потом аналогично введённому слову на английском, происходят проверки его перевода и затем проверяется что путь до картинки указан.

Для оценки сложности процедуры по метрике Холстеда, необходимо посчитать количество общих и уникальных операторов и операндов. Операторы представлены в таблице 6, операнды – в таблице 10.

Таблица 10 – Операторы процедуры

Операторы	Число вхождений
1	2
1	2
string.IsNullOrEmpty()	3
Debug.Log()	5
;	11
if	5
return	6
{	6
}	6
!	2
Regex.IsMatch()	2
n1 = 9	N1 = 46

Таблица 11 – Операнды процедуры

Операнды	Число вхождений
englishWord	3
russianWord	3
pathToImage	2
RUSSIAN_PATTERN	1
ENGlISH_PATTERN	1
"Введите слово на английском"	1
"Введенное слово некорректно"	2
"Введите перевод слова на русском"	1
"Картинка для слова не была добавлена"	1
True	1
False	5
n2 = 11	N2 = 21

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 46 + 21 = 67.$$

где N1 – число вхождений операторов;

N2 – число вхождений операндов.

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 9 + 11 = 20.$$

где n1 – количество операторов;

n2 – количество операндов.

Объем программы рассчитывается по формуле:

$$HV = N * \log2(n) = 67 * 4,3 = 288.1.$$

где N – длина функции;

n – словарь функции.

Потенциальный объем программы

$$V^* = (2 + n`) * \log2(2 + n`) = 8.$$

где n` – количество уникальных операндов.

Теоретический уровень программы рассчитывается по формуле:

$$L` = V^* / HV = 8/288.1 = 0,0277.$$

где V* – потенциальный объем функции;

HV – объем функции.

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (9 / 2) * (21 / 11) = 8,59.$$

где n1 – количество операторов;

N2 – число вхождений операндов;

n2 – количество операндов.

Сложность функции приблизительно равна 9, поэтому можно считать её простой.

5.3 Автоматизированное тестирование на проверку дубликатов

Автоматизированное тестирование – это метод тестирования программного обеспечения, который выполняется с использованием специальных программных средств, которые, в свою очередь необходимы для выполнения набора тестовых примеров.

Был написан тест для проверки на дубликаты уже записанных слов и представлен в листинге 2.

Листинг 2 – Тест для проверки на дубликаты

```
private void CheckDuplicates()
{
XmlElement rootElement = xmlDocument.DocumentElement!;
foreach (XmlNode node in rootElement) {
bool repeatedNotFound = true;
foreach (XmlElement childNode in node) {
int countRepeated = 0;
foreach (XmlElement childNode1 in node) {
if (childNode.InnerText != childNode1.InnerText) {
continue;
}
countRepeated++;
}

if (countRepeated > 1) {
repeatedNotFound = false;
Debug.Log($"Найден дубликат:{childNode.InnerText}");
}
}

Debug.Log($"Тест завершён");
if (repeatedNotFound) {
```

```
Debug.Log($"Всё хорошо. Дубликатов нет.");
}
}
}
```

Результат работы теста представлен на рисунке 13.

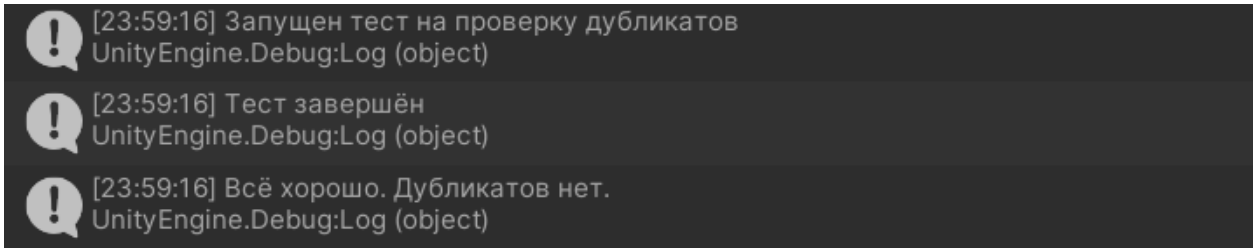


Рисунок 13 – Результат тестирования функции по валидации

В результате запуска можно увидеть, что тестирование прошло успешно.

Заключение

В результате выполнения выпускной квалификационной работы было разработано мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме.

На этапе анализа требований и предметной области были подобраны и проанализированы аналоги. В результате анализа была проведена сравнительная характеристика, в ходе которой были выявлены основные преимущества и недостатки аналогов, на основе которых были определены требования, которым должно соответствовать разрабатываемое мобильное приложение.

Был осуществлен функционально-стоимостной анализ, в ходе которого был определен бюджет и затраты разработки. В результате анализа было определено, что всего разработка приложения займет 47 дней, а затраты на реализацию системы составят 136 100 рублей.

Было проведено проектирование мобильного приложения, заключающееся в проектировании структуры приложения и создания макета пользовательского интерфейса приложения.

Далее была проведена разработка системы, используя язык программирования C# и кроссплатформенную среду разработки Unity. Было проведено тестирование методами черного ящика, метрикой Холстеда и автоматизированное тестирование, показавшее, что программа работает корректно.

Таким образом, в результате выполнения выпускной квалификационной работы было разработано мобильное приложение для запоминания английских слов с применением системы Лейтнера в игровой форме.

Список использованных источников

1. Система Лейтнера // Википедия. [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Система_Лейтнера (дата обращения: 10.06.2023).

2. Система Лейтнера: 5 шагов, позволяющих выучить что угодно // 4Brain. [Электронный ресурс] URL: https://4brain.ru/blog/sistema-lejtnera/ (дата обращения: 10.06.2023).

3. Обзор Brainscape. // All In One Person. [Электронный ресурс] URL: https://blog.themarfa.name/obzor-brainscape-profiessionalnaia-platforma/ (дата обращения: 10.06.2023).

4. Quizlet: что это и как работает? // Skyteach. [Электронный ресурс] URL: https://skyteach.ru/2019/12/04/quizlet-chto-eto-i-kak-rabotaet/ (дата обращения: 10.06.2023).

5. Элементы графической нотации диаграммы вариантов использования // НОУ ИНТУИТ [Электронный ресурс]. URL: https://intuit.ru/studies/courses/32/32/lecture/1004 (дата обращения: 10.06.2023).

6. Star UML. Руководство пользователя // StarUML. [Электронный ресурс] URL: https://staruml.sourceforge.net/docs/user-guide(ru)/user-guide.pdf#:~:text=StarUML%20™%20%20программный%20инструмент,архитектура)%2C%20реализуя%20концепцию%20профилей%20UML (дата обращения: 10.06.2023).

7. Unity // Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/unity/. (дата обращения: 10.06.2023).

8. Разбираем диаграмму Ганта – инструмент, который должен знать каждый менеджер // Skillbox. [Электронный ресурс] URL: https://skillbox.ru/media/management/razbiraem-diagrammu-ganta-instrument-kotoryy-dolzhen-znat-kazhdyy-menedzher/ (дата обращения: 10.06.2023).

9. Dependency Injection: Принцип. Паттерн. Контейнер // Calabonga SOFT. [Электронный ресурс] URL: https://www.calabonga.net/blog/post/dependency-injection-principle-pattern-container (дата обращения: 10.06.2023).

10. Как построить диаграмму классов UML // Техноблог. [Электронный ресурс] URL: https://itonboard.ru/analysis/705_diagramma_klassov_uml_class_diagram_polnoe_rukovodstvo_s_primerami/ (дата обращения: 10.06.2023).

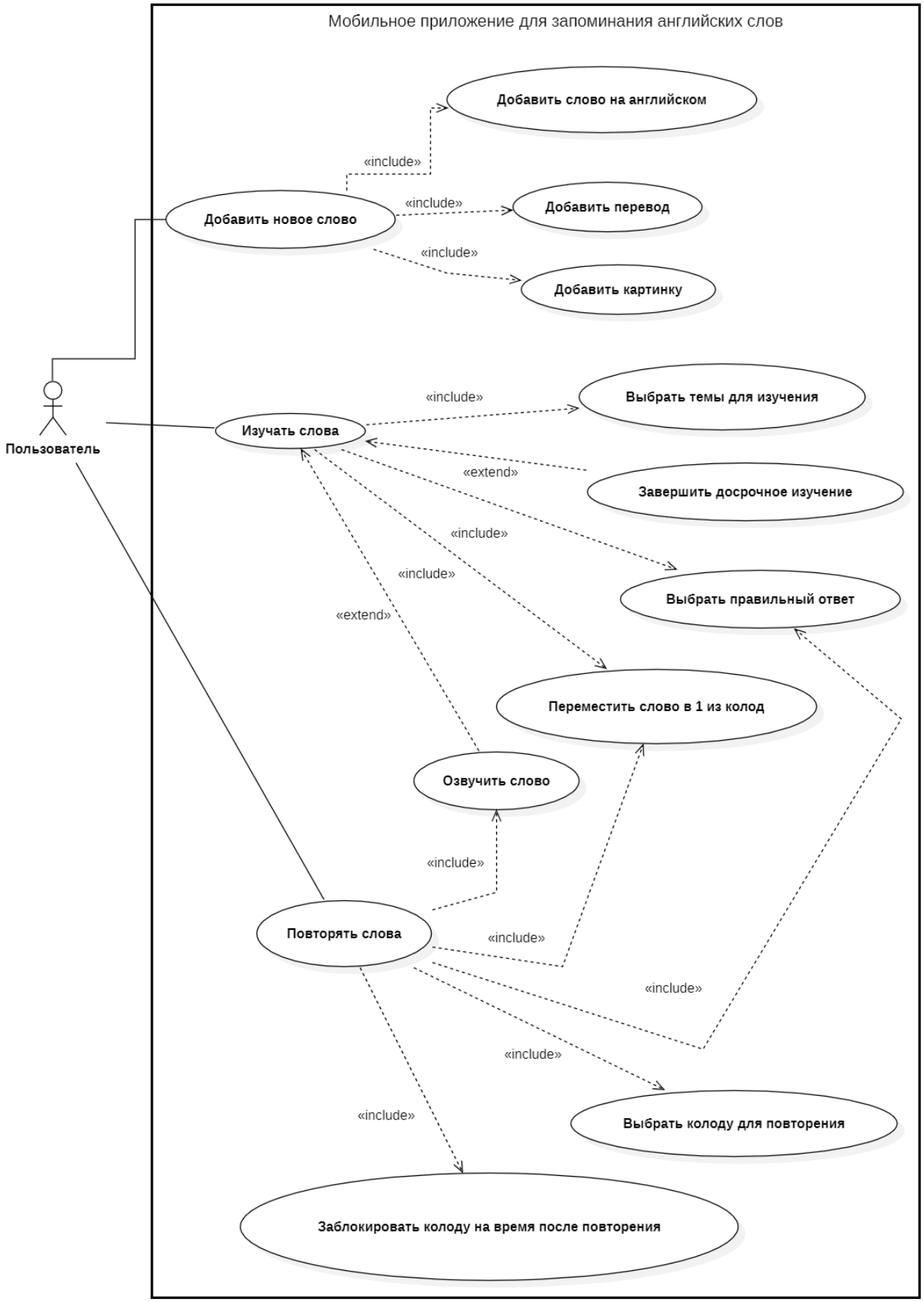
11. Элементы графической нотации диаграммы компонентов // НОУ ИНТУИТ. [Электронный ресурс] URL: https://intuit.ru/studies/courses/32/32/lecture/1022 (дата обращения: 10.06.2023).

12. Особенности тестирования «черного ящика» // [Электронный ресурс] / Лаборатория качества. URL: https://quality-lab.ru/blog/key-principles-of-black-box-testing/ (дата обращения: 05.06.2023).

13. Тестирование методом черного ящика // Timofeev.ru [Электронный ресурс] URL: https://temofeev.ru/info/articles/testirovanie-metodom-chernogo-yashchika/ (дата обращения: 05.06.2023).

14. Общие сведения о программном обеспечении // НОУ ИНТУИТ [Электронный ресурс] URL: https://intuit.ru/studies/professional_skill_improvements/17031/courses/874/lecture/14289?page=2 (дата обращения: 05.06.2023).

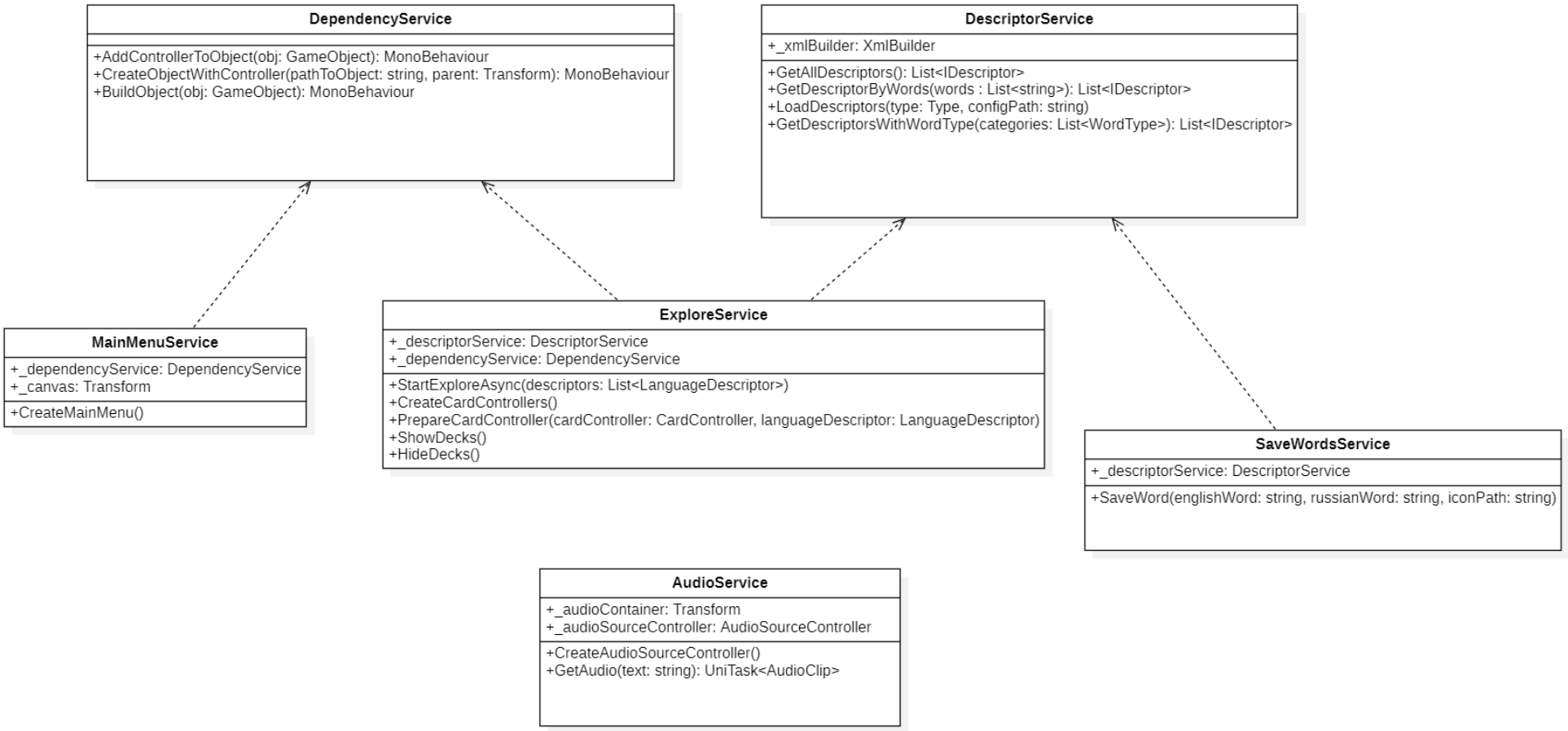
ПРИЛОЖЕНИЕ А
ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ
(рекомендуемое)


Мобильное приложение для запоминания английских слов
Добавить слово на английском
«include»
Добавить новое слово
«include»
Добавить перевод
«include»
Добавить картинку
Пользователь
«include»
Выбрать темы для изучения
Изучать слова
«extend»
Завершить досрочное изучение
«include»
«include»
Выбрать правильный ответ
«extend»
Переместить слово в 1 из колод
Озвучить слово
«include»
Повторять слова
«include»
«include»
«include»
Выбрать колоду для повторения
«include»
Заблокировать колоду на время после повторения

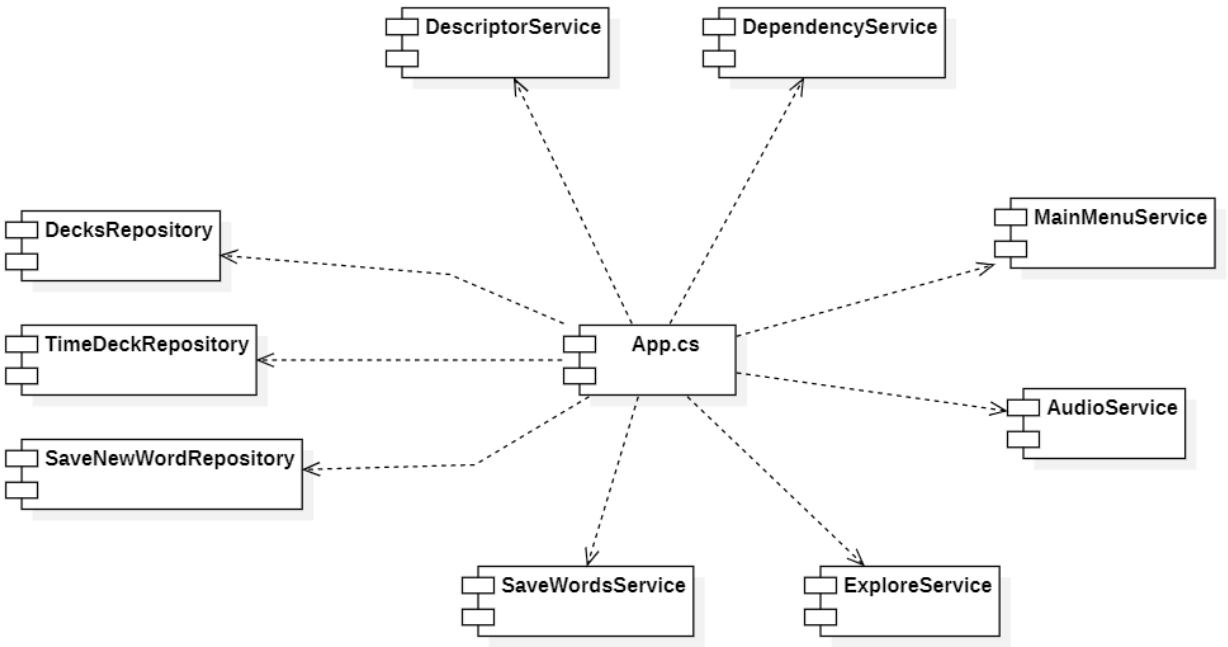
ПРИЛОЖЕНИЕ Б

ДИАГРАММА КЛАССОВ

(рекомендуемое)


DependencyService
+AddControllerToObject(obj: GameObject): MonoBehaviour
+CreateObjectWithController(pathToObject: string, parent: Transform): MonoBehaviour
+BuildObject(obj: GameObject): MonoBehaviour
DescriptorService
+_xmlBuilder: XmlBuilder
+GetAllDescriptors(): List<IDescriptor>
+GetDescriptorByWords(words : List<string>): List<IDescriptor>
+LoadDescriptors(type: Type, configPath: string)
+GetDescriptorsWithWordType(categories: List<WordType>): List<IDescriptor>
MainMenuService
+_dependencyService: DependencyService
+_canvas: Transform
+CreateMainMenu()
ExploreService
+_descriptorService: DescriptorService
+_dependencyService: DependencyService
+StartExploreAsync(descriptors: List<LanguageDescriptor>)
+CreateCardControllers()
+PrepareCardController(cardController: CardController, languageDescriptor: LanguageDescriptor)
+ShowDecks()
+HideDecks()
SaveWordsService
+_descriptorService: DescriptorService
+SaveWord(englishWord: string, russianWord: string, iconPath: string)
AudioService
+_audioContainer: Transform
+_audioSourceController: AudioSourceController
+CreateAudioSourceController()
+GetAudio(text: string): UniTask<AudioClip>

ПРИЛОЖЕНИЕ В
ДИАГРАММА КОМПОНЕНТОВ
(рекомендуемое)


DescriptorService
DependencyService
DecksRepository
MainMenuService
TimeDeckRepository
App.cs
AudioService
SaveNewWordRepository
SaveWordsService
ExploreService

ПРИЛОЖЕНИЕ Г
ЛИСТИНГ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг Г.1 – Компонент App.cs

```
using Audio.Service;
using Core.IoC.Container;
using Decks.Model;
using Decks.Repository;
using Dependency.Service;
using Descriptors.Service;
using Explore.Service;
using SaveWords.Model;
using SaveWords.Repositroy;
using SaveWords.Service;
using Time;
using UI.MainMenu;
using UnityEngine;

namespace Application
{
    public class App : MonoBehaviour
    {
        public static IocContainer IoCContainer { get; private set; } = null!;
        private void Awake()
        {
            Debug.Log("App awake");
            IoCContainer = gameObject.AddComponent<IocContainer>();

            IoCContainer.RegisterSingletonClass<DecksRepository>();
            IoCContainer.RegisterSingletonClass<TimeDeckRepository>();
            IoCContainer.RegisterSingletonClass<SaveNewWordRepository>();

IoCContainer.RegisterSingletonMonoBehaviourClass<DescriptorService>();

IoCContainer.RegisterSingletonMonoBehaviourClass<DependencyService>();
            IoCContainer.RegisterSingletonMonoBehaviourClass<MainMenuService>();
            IoCContainer.RegisterSingletonMonoBehaviourClass<AudioService>();
            IoCContainer.RegisterSingletonMonoBehaviourClass<ExploreService>();

IoCContainer.RegisterSingletonMonoBehaviourClass<SaveWordsService>();

            DecksRepository decksRepository =
IoCContainer.RequireInstance<DecksRepository>();
            TimeDeckRepository timeDeckRepository =
IoCContainer.RequireInstance<TimeDeckRepository>();
            SaveNewWordRepository saveNewWordRepository =
IoCContainer.RequireInstance<SaveNewWordRepository>();



            if (decksRepository.Get() == null) {
                decksRepository.Set(new DecksModel());
            }

            if (timeDeckRepository.Get() == null) {
                timeDeckRepository.Set(new TimeDeckModel());
            }

            if (saveNewWordRepository.Get() == null) {
                saveNewWordRepository.Set(new NewWordsModel());
            }
        }
    }
```

```
}
```

Листинг Г.2 – Компонент DecksModel.cs

```
using System.Collections.Generic;

namespace Decks.Model
{
    public class DecksModel
    {
        private List<string> _decks1;
        private List<string> _decks2;
        private List<string> _decks3;

        public DecksModel()
        {
            _decks1 = new List<string>();
            _decks2 = new List<string>();
            _decks3 = new List<string>();
        }

        public void AddWordInDecks1(string word)
        {
            if (_decks2.Contains(word)) {
                RemoveWordFromDecks2(word);
            }
            if (_decks3.Contains(word)) {
                RemoveWordFromDecks3(word);
            }
            if (_decks1.Contains(word)) {
                return;
            }

            _decks1.Add(word);
        }

        public void AddWordInDecks2(string word)
        {
            if (_decks1.Contains(word)) {
                RemoveWordFromDecks1(word);
            }
            if (_decks3.Contains(word)) {
                RemoveWordFromDecks3(word);
            }
            if (_decks2.Contains(word)) {
                return;
            }

            _decks2.Add(word);
        }

        public void AddWordInDecks3(string word)
        {
            if (_decks1.Contains(word)) {
                RemoveWordFromDecks1(word);
            }
            if (_decks2.Contains(word)) {
                RemoveWordFromDecks2(word);
            }
            if (_decks3.Contains(word)) {
                return;
            }
```

```
            _decks3.Add(word);
        }

        public void RemoveWordFromDecks1(string word)
        {
            if (!_decks1.Contains(word)) {
                return;
            }

            _decks1.Remove(word);
        }

        public void RemoveWordFromDecks2(string word)
        {
            if (!_decks2.Contains(word)) {
                return;
            }

            _decks2.Remove(word);
        }

        public void RemoveWordFromDecks3(string word)
        {
            if (!_decks3.Contains(word)) {
                return;
            }

            _decks3.Remove(word);
        }

        public bool ExistOnDecks(string word)
        {
            return Decks1.Contains(word) || Decks2.Contains(word) ||
Decks3.Contains(word);
        }

        public List<string> Decks1
        {
            get => _decks1;
        }

        public List<string> Decks2
        {
            get => _decks2;
        }
        public List<string> Decks3
        {
            get => _decks3;
        }
    }
}
```

Листинг Г.3 – Компонент TimeDeckModel.cs

```
using System;

namespace Time
{
    public class TimeDeckModel
    {
        public TimeDeckModel()
        {
            DateTimeDeck1 = new DateTime();
            DateTimeDeck2 = new DateTime();
            DateTimeDeck3 = new DateTime();
        }

        public DateTime DateTimeDeck1 { get; set; }
        public DateTime DateTimeDeck2 { get; set; }
        public DateTime DateTimeDeck3 { get; set; }
    }
}
```

Листинг Г.4 – Компонент NewWordsModel.cs

```
using System.Collections.Generic;

namespace SaveWords.Model
{
    public class NewWordsModel
    {
        private List<NewWords> _newWords;

        public NewWordsModel()
        {
            _newWords = new List<NewWords>();
        }

        public void AddNewWord(NewWords newWords)
        {
            if (_newWords.Contains(newWords)) {
                return;
            }

            _newWords.Add(newWords);
        }

        public List<NewWords> NewWordsList
        {
            get => _newWords;
        }
    }
}
```

Листинг Г.5 – Компонент DescriptorService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml;
using Core.IoC.AttributeInject;
```

```
using Core.XmlReader.Config;
using Core.XmlReader.Service;
using Descriptors.Enumer;
using Descriptors.Interface;
using Descriptors.Model;
using SaveWords.Model;
using SaveWords.Repositroy;
using UnityEngine;
using Utils.Constants;

namespace Descriptors.Service
{
    public class DescriptorService : MonoBehaviour
    {
        private static XmlBuilder _xmlBuilder = new();

        private Dictionary<Type, List<IDescriptor>> _descriptors = new();

        [Dependence]
        private SaveNewWordRepository _saveNewWordRepository;

        private void Start()
        {
            Debug.Log("DescriptorService start");
            LoadDescriptors(typeof(LanguageDescriptor),
GameConstants.LANGUAGE_CONFIG);
            Debug.Log("Descriptors loaded");
        }

        public List<T> GetAllDescriptors<T>() where T : class
        {
            Type type = typeof(T);
            if (!_descriptors.ContainsKey(type)) {
                throw new NullReferenceException($"Descriptors with type not
found. Type={type.Name}");
            }

            return _descriptors[type].Cast<T>().ToList() ?? throw new
InvalidCastException($"Dont cast descriptors. Type={type}");
        }

        public List<LanguageDescriptor> GetDescriptorByWords(List<string> words)
        {
            List<LanguageDescriptor> result = new List<LanguageDescriptor>();
            List<LanguageDescriptor> languageDescriptors =
GetAllDescriptors<LanguageDescriptor>();
            foreach (LanguageDescriptor languageDescriptor in
languageDescriptors)
            {
                if (!words.Contains(languageDescriptor.EnglishWord)) {
                    continue;
                }

                result.Add(languageDescriptor);
            }

            return result;
        }

        public List<LanguageDescriptor>
GetDescriptorsWithWordType(List<WordType> categories)
        {
            List<LanguageDescriptor> result = new List<LanguageDescriptor>();
```

```
            List<LanguageDescriptor> languageDescriptors =
GetAllDescriptors<LanguageDescriptor>();
            foreach (LanguageDescriptor languageDescriptor in
languageDescriptors)
            {
                if (!categories.Contains(languageDescriptor.WordType)) {
                    continue;
                }

                result.Add(languageDescriptor);
            }

            return result;
        }

        public void LoadDescriptors(Type type, string configPath)
        {
            _descriptors = new();
            List<IDescriptor> descriptors = new();
            XmlDocument config = _xmlBuilder.CreateXmlDocument(configPath);
            List<Configuration> configurations =
_xmlBuilder.LoadConfiguration(config);

            foreach (Configuration item in configurations) {
                IDescriptor descriptor = (IDescriptor)
Activator.CreateInstance(type);
                descriptor.SetData(item);
                descriptors.Add(descriptor);
            }

            NewWordsModel newWordsModel = _saveNewWordRepository.Get();
            foreach (NewWords newWords in newWordsModel.NewWordsList)
            {
                IDescriptor descriptor = (IDescriptor)
Activator.CreateInstance(type);
                descriptor.SetData(newWords.EnglishWord, newWords.RussianWord,
newWords.ImagePath);
                descriptors.Add(descriptor);
            }

            _descriptors.Add(type, descriptors);
        }
    }
}
```

Листинг Г.6 – Компонент DescriptorService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml;
using Core.IoC.AttributeInject;
using Core.XmlReader.Config;
using Core.XmlReader.Service;
using Descriptors.Enumer;
using Descriptors.Interface;
using Descriptors.Model;
using SaveWords.Model;
using SaveWords.Repositroy;
using UnityEngine;
using Utils.Constants;
```

```
namespace Descriptors.Service
{
    public class DescriptorService : MonoBehaviour
    {
        private static XmlBuilder _xmlBuilder = new();

        private Dictionary<Type, List<IDescriptor>> _descriptors = new();

        [Dependence]
        private SaveNewWordRepository _saveNewWordRepository;

        private void Start()
        {
            Debug.Log("DescriptorService start");
            LoadDescriptors(typeof(LanguageDescriptor),
GameConstants.LANGUAGE_CONFIG);
            Debug.Log("Descriptors loaded");
        }

        public List<T> GetAllDescriptors<T>() where T : class
        {
            Type type = typeof(T);
            if (!_descriptors.ContainsKey(type)) {
                throw new NullReferenceException($"Descriptors with type not
found. Type={type.Name}");
            }

            return _descriptors[type].Cast<T>().ToList() ?? throw new
InvalidCastException($"Dont cast descriptors. Type={type}");
        }

        public List<LanguageDescriptor> GetDescriptorByWords(List<string> words)
        {
            List<LanguageDescriptor> result = new List<LanguageDescriptor>();
            List<LanguageDescriptor> languageDescriptors =
GetAllDescriptors<LanguageDescriptor>();
            foreach (LanguageDescriptor languageDescriptor in
languageDescriptors)
            {
                if (!words.Contains(languageDescriptor.EnglishWord)) {
                    continue;
                }

                result.Add(languageDescriptor);
            }

            return result;
        }

        public List<LanguageDescriptor>
GetDescriptorsWithWordType(List<WordType> categories)
        {
            List<LanguageDescriptor> result = new List<LanguageDescriptor>();
            List<LanguageDescriptor> languageDescriptors =
GetAllDescriptors<LanguageDescriptor>();
            foreach (LanguageDescriptor languageDescriptor in
languageDescriptors)
            {
                if (!categories.Contains(languageDescriptor.WordType)) {
                    continue;
                }

                result.Add(languageDescriptor);
            }
```

```
            return result;
        }

        public void LoadDescriptors(Type type, string configPath)
        {
            _descriptors = new();
            List<IDescriptor> descriptors = new();
            XmlDocument config = _xmlBuilder.CreateXmlDocument(configPath);
            List<Configuration> configurations =
_xmlBuilder.LoadConfiguration(config);

            foreach (Configuration item in configurations) {
                IDescriptor descriptor = (IDescriptor)
Activator.CreateInstance(type);
                descriptor.SetData(item);
                descriptors.Add(descriptor);
            }

            NewWordsModel newWordsModel = _saveNewWordRepository.Get();
            foreach (NewWords newWords in newWordsModel.NewWordsList)
            {
                IDescriptor descriptor = (IDescriptor)
Activator.CreateInstance(type);
                descriptor.SetData(newWords.EnglishWord, newWords.RussianWord,
newWords.ImagePath);
                descriptors.Add(descriptor);
            }

            _descriptors.Add(type, descriptors);
        }
    }
}
```

Листинг Г.7 – Компонент DependencyService.cs

```
using Application;
using UnityEngine;

namespace Dependency.Service
{
    public class DependencyService : MonoBehaviour
    {
        public T AddControllerToObject<T>(GameObject obj) where T :
MonoBehaviour
        {
            return BuildObject<T>(obj);
        }

        public T CreateObjectWithController<T>(string pathToObject, Transform
parent) where T : MonoBehaviour
        {
            GameObject result =
Instantiate(Resources.Load<GameObject>(pathToObject), parent);
            return BuildObject<T>(result);
        }

        private T BuildObject<T>(GameObject obj) where T : MonoBehaviour
        {
            return App.IoCContainer.AttachControllerToGameObject<T>(obj);
        }
    }
```

```
}
```

Листинг Г.8 – Компонент MainMenuService.cs

```
using Core.IoC.AttributeInject;
using Dependency.Service;
using UI.MainMenu.Controller;
using UnityEngine;
using Utils.Constants;

namespace UI.MainMenu
{
    public class MainMenuService : MonoBehaviour
    {
        [Dependence]
        private DependencyService _dependencyService;

        private Transform _canvas;

        private void Start()
        {
            Debug.Log("UIService start");
            _canvas = gameObject.GetComponentInChildren<Canvas>().transform;
            CreateMainMenu();
        }

        private void CreateMainMenu()
        {
            MainMenuController mainMenuController =
_dependencyService.CreateObjectWithController<MainMenuController>(GameConstants.
MAIN_MENU, _canvas);
            mainMenuController.transform.SetAsFirstSibling();
        }

        public Transform Canvas => _canvas;
        public float ScaleFactor => _canvas.GetComponent<Canvas>().scaleFactor;
    }
}
```

Листинг Г.9 – Компонент AudioService.cs

```
using Audio.Controller;
using Cysharp.Threading.Tasks;
using UnityEngine;
using UnityEngine.Networking;
using Utils.Constants;

namespace Audio.Service
{
    public class AudioService : MonoBehaviour
    {
        private Transform _audioContainer;
        private AudioSourceController _audioSourceController;

        private void Start()
        {
            _audioContainer =
GameObject.Find(GameConstants.AUDIO_CONTAINER).transform;
            CreateAudioSourceController();
        }
```

```
        public async UniTask SoundTheWord(string text)
        {
            AudioClip audioClip = await GetAudio(text);
            _audioSourceController.PlayOneShot(audioClip);
        }

        private void CreateAudioSourceController()
        {
            GameObject audioSourceObject =
Resources.Load<GameObject>(GameConstants.SOUND_SOURCE);
            GameObject audioSourceInstance = Instantiate(audioSourceObject,
_audioContainer, false);
            _audioSourceController =
audioSourceInstance.AddComponent<AudioSourceController>();
        }

        private async UniTask<AudioClip> GetAudio(string text)
        {
            string endedUrl;
            if (SystemInfo.deviceType == DeviceType.Desktop) {
                endedUrl = GameConstants.URL_GOOGLE_TRANSLATE_PC + text + "&tl="
+ "en";
            }
            else {
                endedUrl = GameConstants.URL_GOOGLE_TRANSLATE_ANDROID + text +
"&tl=" + "en";
            }
            using (UnityWebRequest webRequest =
UnityWebRequestMultimedia.GetAudioClip(endedUrl, AudioType.MPEG))
            {
                await webRequest.SendWebRequest().ToUniTask();
                if (webRequest.result == UnityWebRequest.Result.Success)
                {
                    return DownloadHandlerAudioClip.GetContent(webRequest);
                }
            }

            Debug.LogError($"Dont load audioClip. Text={text}. Url={endedUrl}");
            return null;
        }
    }
}
```

Листинг Г.10 – Компонент ExploreService.cs

```
using System.Collections.Generic;
using System.Linq;
using Core.Events.System;
using Core.IoC.AttributeInject;
using Cysharp.Threading.Tasks;
using Decks.Model;
using Decks.Repository;
using Dependency.Service;
using Descriptors.Enumer;
using Descriptors.Model;
using Descriptors.Service;
using UI.Cards.Controller;
using UI.MainMenu.Events;
using UnityEngine;
using Utils.Constants;
using Random = System.Random;
```

```
namespace Explore.Service
{
    public class ExploreService : EventsComponent
    {
        [Dependence]
        private DescriptorService _descriptorService;
        [Dependence]
        private DependencyService _dependencyService;
        [Dependence]
        private DecksRepository _decksRepository;

        private List<CardController> _cardControllers = new();
        private Transform _cardContainer;
        private float _scaleFactor;
        private Random _random = new();

        public bool Canceled { get; set; }
        public bool ChoiceIsMade { get; set; }

        private void Start()
        {
            _cardContainer =
GameObject.Find(GameConstants.CARD_CONTAINER).transform;
            _scaleFactor =
gameObject.GetComponentInChildren<Canvas>().scaleFactor;
            CreateCardControllers();
        }

        public async UniTask StartExploreAsync(List<string> words)
        {
            List<LanguageDescriptor> descriptors =
_descriptorService.GetDescriptorByWords(words);
            if (descriptors.Count <= 0) {
                //todo мб какой-нибудь вывод
                Invoke(new StopExploreEvent());
                return;
            }

            await StartExploreAsync(descriptors);
        }

        public async UniTaskVoid StartExploreAsync(List<WordType>
allowedCategories)
        {
            List<LanguageDescriptor> allDescriptors =
_descriptorService.GetDescriptorsWithWordType(allowedCategories);
            List<LanguageDescriptor> descriptors =
ExcludeExistCards(allDescriptors);

            if (descriptors.Count == 0) {
                //todo мб какой-нибудь вывод
                Invoke(new StopExploreEvent());
                return;
            }

            await StartExploreAsync(descriptors);
            DecksModel decksModel = _decksRepository.Get();
            foreach (WordType allowedCategory in allowedCategories)
            {
                List<LanguageDescriptor> descriptorsWithWordType =
_descriptorService.GetDescriptorsWithWordType(new List<WordType>()
{allowedCategory});
                bool existOnDecks = true;
```

```
                foreach (LanguageDescriptor languageDescriptor in
descriptorsWithWordType)
                {
                    if (decksModel.ExistOnDecks(languageDescriptor.EnglishWord))
{
                        continue;
                    }

                    existOnDecks = false;
                    break;
                }

                if (!existOnDecks) {
                    continue;
                }

                Invoke(new UnlockDecksAfterExplore());
                return;
            }
        }

        public async UniTask StartExploreAsync(List<LanguageDescriptor>
descriptors)
        {
            Canceled = false;
            descriptors = descriptors.OrderBy(v => _random.Next()).ToList();
            int descriptorIndex = 0;

            while (!Canceled)
            {
                if (descriptorIndex >= descriptors.Count) {
                    Canceled = true;
                    break;
                }

                LanguageDescriptor descriptor = descriptors[descriptorIndex];
                CardController freeCardController = GetFreeCardController();
                PrepareCardController(freeCardController, descriptor);
                ShowChoiceButtons(descriptor, descriptors);
                Invoke(new OnNewCardEvent(descriptorIndex+1,
descriptors.Count));
                await UniTask.WaitWhile(() => !ChoiceIsMade);
                await UniTask.Delay(500);
                ChoiceIsMade = false;
                HideChoiceButtons();
                if (Canceled) {
                    freeCardController.gameObject.SetActive(false);
                    break;
                }
                freeCardController.EnableDragAndDrop();
                ShowDecks();
                await UniTask.WaitWhile(() => !ChoiceIsMade);
                ChoiceIsMade = false;
                HideDecks();
                freeCardController.gameObject.SetActive(false);
                descriptorIndex++;
            }

            Invoke(new StopExploreEvent());
        }

        private void CreateCardControllers()
        {
            for (int i = 0; i < GameConstants.COUNT_STARTED_CARDS; i++)
```

```
            {
                CardController cardController =
_dependencyService.CreateObjectWithController<CardController>(GameConstants.CARD
, _cardContainer);
                _cardControllers.Add(cardController);
            }
        }

        private void PrepareCardController(CardController cardController,
LanguageDescriptor languageDescriptor)
        {
            cardController.gameObject.SetActive(true);
            cardController.Refresh(languageDescriptor.EnglishWord,
languageDescriptor.Image, _scaleFactor, languageDescriptor.NeedShowText);
            cardController.DisableDragAndDrop();
        }

        private void ShowChoiceButtons(LanguageDescriptor currentCardDescriptor,
List<LanguageDescriptor> otherDescriptors)
        {
            if (otherDescriptors.Count <= 3) {
                otherDescriptors =
_descriptorService.GetAllDescriptors<LanguageDescriptor>();
            }
            List<string> otherWords = new List<string>();
            Random random = new Random();
            bool ready = false;
            while (!ready)
            {
                int randomIndex = random.Next(1, otherDescriptors.Count);
                LanguageDescriptor randomDescriptor =
otherDescriptors[randomIndex];
                if (otherWords.Contains(randomDescriptor.RussianWord) ||
randomDescriptor.RussianWord == currentCardDescriptor.RussianWord) {
                    continue;
                }

                otherWords.Add(randomDescriptor.RussianWord);
                if (otherWords.Count < 3) {
                    continue;
                }

                ready = true;
            }

            Invoke(new
ShowChoiceButtonsEvent(otherWords,currentCardDescriptor.RussianWord,
currentCardDescriptor.EnglishWord));
        }

        private void HideChoiceButtons()
        {
            Invoke(new HideChoiceButtonsEvent());
        }

        private void ShowDecks()
        {
            Invoke(new ShowDecksEvent());
        }

        private void HideDecks()
        {
            Invoke(new HideDecksEvent());
        }
```

```
        private List<LanguageDescriptor>
ExcludeExistCards(List<LanguageDescriptor> descriptors)
        {
            List<LanguageDescriptor> result = new List<LanguageDescriptor>();
            DecksModel decksModel = _decksRepository.Require();
            foreach (LanguageDescriptor languageDescriptor in descriptors)
            {
                if (decksModel.ExistOnDecks(languageDescriptor.EnglishWord)) {
                    continue;
                }

                result.Add(languageDescriptor);
            }

            return result;
        }

        private CardController GetFreeCardController()
        {
            return _cardControllers.First(cc => cc.Free);
        }
    }
}
```

Листинг Г.11 – Компонент SaveWordsService.cs

```
using System.Xml;
using Core.IoC.AttributeInject;
using Core.XmlReader.Service;
using Descriptors.Model;
using Descriptors.Service;
using SaveWords.Model;
using SaveWords.Repositroy;
using UnityEngine;
using Utils.Constants;

namespace SaveWords.Service
{
    public class SaveWordsService : MonoBehaviour
    {
        private XmlBuilder _xmlBuilder = new();

        [Dependence]
        private DescriptorService _descriptorService;
        [Dependence]
        private SaveNewWordRepository _saveNewWordRepository;

        public void SaveWord(string englishWord, string russianWord, string
iconPath)
        {
            XmlDocument xmlDocument =
_xmlBuilder.CreateXmlDocument(GameConstants.LANGUAGE_CONFIG);
            XmlElement xmlElementMain = xmlDocument.CreateElement("word");

            XmlElement xmlElementEnglishWord =
xmlDocument.CreateElement("englishWord");
            XmlElement xmlElementRussianWord =
xmlDocument.CreateElement("russianWord");
            XmlElement xmlElementIconPath = xmlDocument.CreateElement("image");
            XmlElement xmlElementWordType =
xmlDocument.CreateElement("wordType");
```

```
            XmlElement xmlElementNeedShowText =
xmlDocument.CreateElement("needShowText");

            xmlElementEnglishWord.InnerText = englishWord;
            xmlElementRussianWord.InnerText = russianWord;
            xmlElementIconPath.InnerText = iconPath;
            xmlElementWordType.InnerText = "other";
            xmlElementNeedShowText.InnerText = "true";

            xmlElementMain.AppendChild(xmlElementEnglishWord);
            xmlElementMain.AppendChild(xmlElementRussianWord);
            xmlElementMain.AppendChild(xmlElementIconPath);
            xmlElementMain.AppendChild(xmlElementWordType);
            xmlElementMain.AppendChild(xmlElementNeedShowText);

            xmlDocument.DocumentElement!.AppendChild(xmlElementMain);
            if (SystemInfo.deviceType == DeviceType.Desktop) {
               string pcPath = UnityEngine.Application.dataPath +
GameConstants.FULL_LANGUAGE_CONFIG_PATH;
               Debug.Log("Path" + $"{pcPath}");
               xmlDocument.Save(pcPath);
            } else {
                NewWordsModel newWordsModel = _saveNewWordRepository.Get();
                NewWords newWords = new NewWords
                {
                    EnglishWord = englishWord,
                    RussianWord = russianWord,
                    ImagePath = iconPath
                };
                newWordsModel.AddNewWord(newWords);
                _saveNewWordRepository.Set(newWordsModel);
            }

            _descriptorService.LoadDescriptors(typeof(LanguageDescriptor),
GameConstants.LANGUAGE_CONFIG);;
        }
    }
}
```

ПРИЛОЖЕНИЕ Д
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

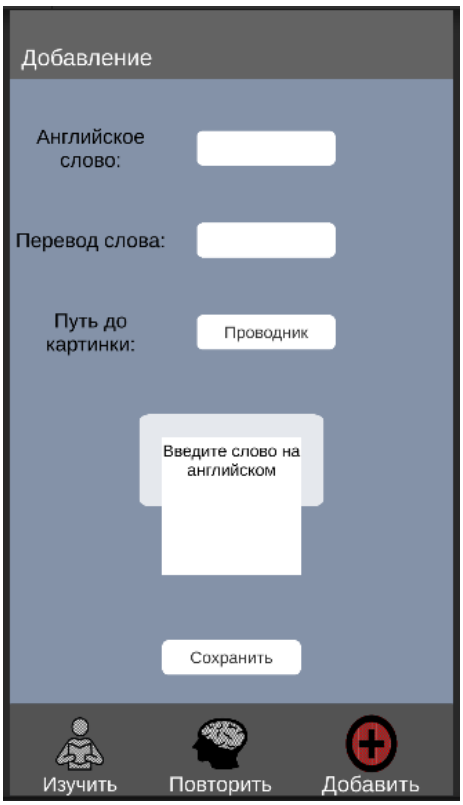


Рисунок Д.1 – Результат теста №1 таблицы решений

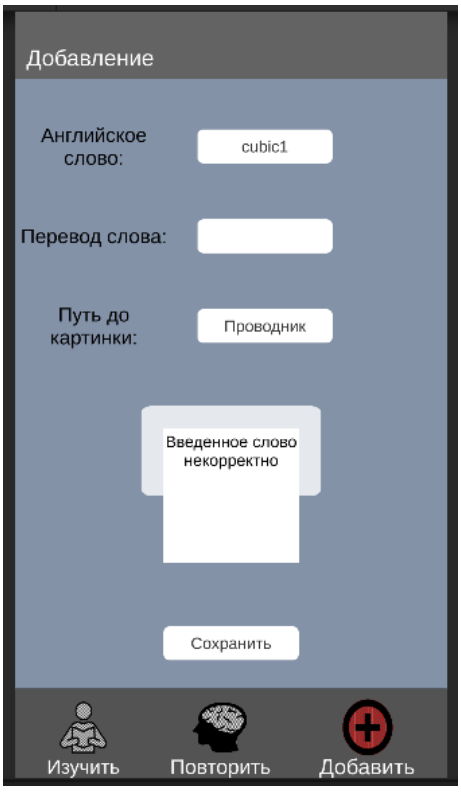


Рисунок Д.2 – Результат теста №2 таблицы решений

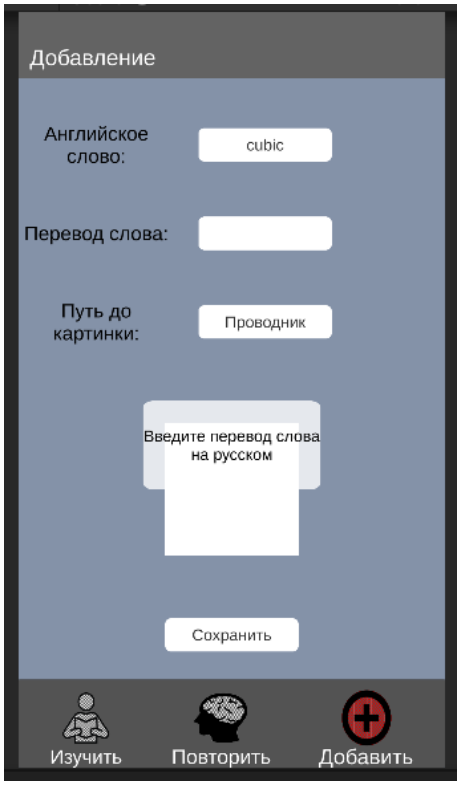


Рисунок Д.3 – Результат теста №3 таблицы решений

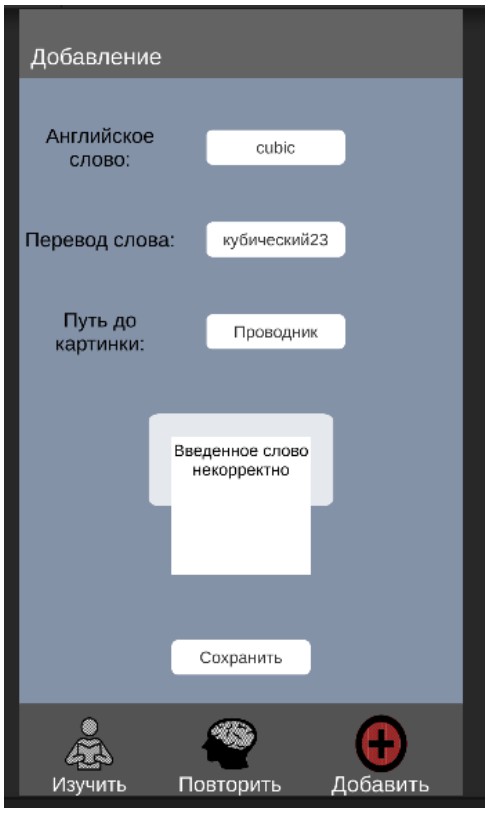


Рисунок Д.4 – Результат теста №4 таблицы решений

Рисунок Д.5 – Результат теста №5 таблицы решений

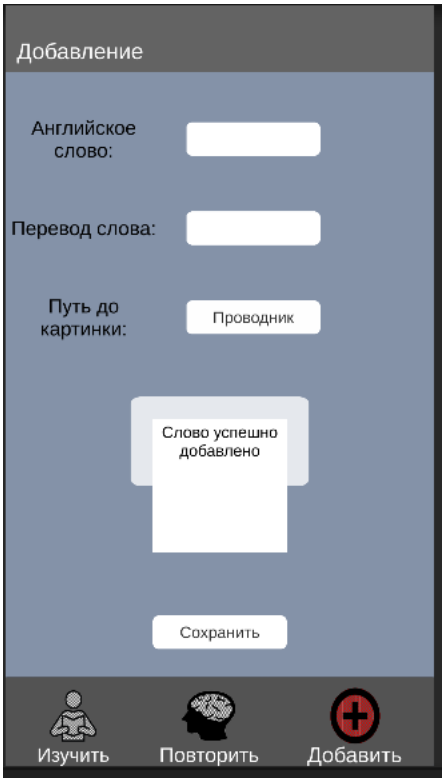


Рисунок Д.6 – Результат теста №6 таблицы решений

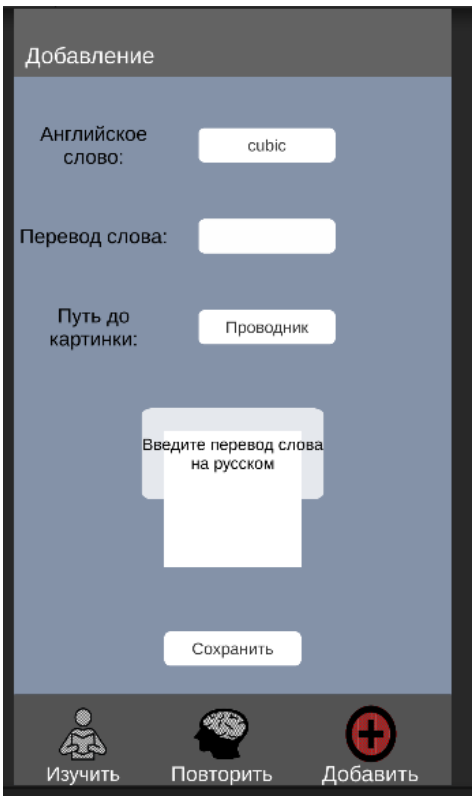


Рисунок Д.7 – Результат теста №7 таблицы решений

Рисунок Д.8 – Результат теста №8 таблицы решений

Рисунок Д.9 – Результат теста №9 таблицы решений