Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Разрешаю допустить к защите

Зав. кафедрой

20 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

НА ТЕМУ

Разработка мобильного приложения для запоминания и тренировки памяти

Студент: Кареева В.В. Руководитель: Королькова Т.В.

Москва 2023 г.

Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра

Системное программирование

«Утверждаю»

Зав. кафедрой

«\_ »\_ 20 г.

(название полностью)

# З А Д А Н И Е

**на выпускную квалификационную работу**

Студенту Кареевой Вере Владимировне гр. БФИ1901

Направление (специальность) 02.03.02 «Фундаментальная информатика и информационные технологии»

Форма выполнения выпускной квалификационной работы бакалаврская работа

(Дипломный проект, дипломная работа, магистерская диссертация, бакалаврская работа)

Тема выпускной квалификационной работы Разработка мобильного приложения для

запоминания и тренировки памяти

Утверждена приказом ректора №87-с от 24.01 2023 г.

1. Исходные данные
   1. Методические рекомендации по подготовке и

защите выпускной квалификационной работы бакалавра 2. ГОСТ 7.32-2001. ССИБД.

ГОСТ 2.105-95. ГОСТ Р7.05-2008

ГОСТ 19.701-90.ЕСПД ГОСТ Р 53801-2010 ГОСТ Р6.30-2003.

1. Kotlin Documentation
2. Интегрированная среда разработки Android Studio Flamingo
   1. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

ВВЕДЕНИЕ

РАЗДЕЛ 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

РАЗДЕЛ 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

РАЗДЕЛ 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЗАКЛЮЧЕНИЕ

Объем работы в % и сроки выполнения по разделам

5% 21.02.2023

35% 19.03.2023

20% 24.04.2023

35% 18.05.2023

5% 19.05.2023

* 1. Вопросы конструктивных разработок
  2. Разработка вопросов по экологии и безопасности жизнедеятельности
  3. Технико-экономическое обоснование (подлежащее расчету)
  4. Перечень графического материала (с точным указанием обязательных чертежей)

1. Блок-схема алгоритма SuperMemo
2. Макет приложения
3. Модель БД
4. Тестирование приложения
5. Консультанты по ВКР (с указанием относящихся к ним разделов проекта):

(подпись) (ФИО)

(подпись) (ФИО)

1. Срок сдачи студентом законченной ВКР: 21.05.2023

Дата выдачи задания: 27.01.2023

Руководитель

(подпись) (ФИО)

нагрузка

(штатная или почасовая)

Задание принял к исполнению

(подпись студента)

Примечание: Настоящее задание прилагается к законченной ВКР

**ОТЗЫВ РУКОВОДИТЕЛЯ**

**на выпускную квалификационную работу студентки группы БФИ1901 КАРЕЕВОЙ ВЕРЫ ВЛАДИМИРОВНЫ**

**на тему**

**«Разработка мобильного приложения для запоминания и тренировки памяти»,**

представленную к защите по направлению 02.03.02 «Фундаментальная информатика и информационные технологии»

**АННОТАЦИЯ**

Выпускная квалификационная работа на тему «Разработка мобильного приложения для запоминания и тренировки памяти» включает в себя 3 раздела, 3 приложения, 89 страниц, 8 рисунков, 5 таблиц и 11 использованных источников.

Цель работы заключается в разработке мобильного приложения, предназначенного для улучшения памяти.

В первом разделе проводится анализ компонентов памяти, ее виды, классификация и фазы, а также подходы к улучшению памяти и некоторые техники, позволяющие облегчить запоминание информации.

Второй раздел включает в себя проектирование приложения, а именно: формирование концепции приложения, определение требований к системе, выбор инструментария.

В третьем разделе содержится описание этапов реализации: создание макета, проектирование базы данных, а также тестирование готового приложения.

Ключевые слова: разработка мобильных приложений, тренировка памяти, интервальное повторение, методы запоминания.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc138006296)

[РАЗДЕЛ 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc138006297)

[1.1 Основные понятия человеческой памяти 9](#_Toc138006298)

[1.1.1 Компоненты памяти по времени удержания информации 9](#_Toc138006299)

[1.1.2 Виды памяти по типу информации 10](#_Toc138006300)

[1.1.3 Классификация по задействованному органу чувств 11](#_Toc138006301)

[1.1.4 Фазы памяти 12](#_Toc138006302)

[1.2 Подходы к улучшению памяти 13](#_Toc138006303)

[1.3 Техники для запоминания 15](#_Toc138006304)

[1.3.1 Метод мнемоники 15](#_Toc138006305)

[1.3.2 Метод локусов 17](#_Toc138006306)

[1.3.3 Метод сторителлинга 20](#_Toc138006307)

[1.3.4 Метод интервального повторения 22](#_Toc138006308)

[1.3.5 Чанкинг-метод 24](#_Toc138006309)

[Вывод 26](#_Toc138006310)

[РАЗДЕЛ 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 27](#_Toc138006311)

[2.1 Изучение существующих реализаций 27](#_Toc138006313)

[2.1.1 Flashcards Maker 27](#_Toc138006314)

[2.1.2 AnkiDroid 28](#_Toc138006315)

[2.1.3 MemoWorld 30](#_Toc138006316)

[2.1.4 Memrise 30](#_Toc138006317)

[2.2 Определение требований к системе 31](#_Toc138006318)

[2.3 Выбор используемых технологий и языка программирования 33](#_Toc138006319)

[2.4 Определение алгоритма запоминания 34](#_Toc138006320)

[2.5 Функциональность приложения 37](#_Toc138006321)

[Вывод 38](#_Toc138006322)

[РАЗДЕЛ 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 39](#_Toc138006323)

[3.1 Создание макета 39](#_Toc138006325)

[3.2 База данных 42](#_Toc138006326)

[3.3 Реализация приложения 45](#_Toc138006327)

[3.3.1 Реализация связи с Firebase 45](#_Toc138006328)

[3.3.2 Реализация запоминания информации 46](#_Toc138006329)

[3.3.3 Реализация упражнений для тренировки памяти 48](#_Toc138006330)

[3.4 Тестирование 49](#_Toc138006331)

[Вывод 51](#_Toc138006332)

[ЗАКЛЮЧЕНИЕ 52](#_Toc138006333)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc138006334)

[ПРИЛОЖЕНИЕ А 55](#_Toc138006335)

[ПРИЛОЖЕНИЕ Б 88](#_Toc138006336)

[ПРИЛОЖЕНИЕ В 92](#_Toc138006337)

**ВВЕДЕНИЕ**

В информационном обществе настоящего времени запоминание информации, и ее воспроизведение, играют важную роль в повседневной жизни, учебе и работе. Тренировка памяти является одним из ключевых методов для поддержания ее эффективности и улучшения производительности в различных сферах жизни.

Современные технологии мобильных устройств позволяют создавать инновационные приложения, которые могут помочь в тренировке памяти. Многие люди сталкиваются с проблемами усвоения информации в повседневной жизни, поэтому разработка приложения для упрощения запоминания является актуальной и востребованной задачей. Такое приложение может помочь пользователям улучшить свою память, повысить производительность и качество жизни.

Объект исследования: средства для запоминания и тренировки памяти.

Предмет исследования: процесс разработки мобильного приложения для запоминания и тренировки памяти.

Целью данной выпускной квалификационной работы является разработка мобильного приложения для запоминания и тренировки памяти. Для достижения данной цели были поставлены следующие задачи:

1. Изучение научных и практических аспектов памяти и ее тренировки.

2. Изучение рынка мобильных приложений для тренировки памяти и анализ существующих приложений.

3. Определение требований к разрабатываемому приложению.

4. Разработка дизайна и интерфейса приложения.

5. Разработка функционала приложения.

6. Тестирование приложения.

Работа содержит введение, 3 раздела, заключение, список использованных источников и приложения.

В первом разделе рассмотрены основные понятия человеческой памяти, ее виды, способы улучшения и расширения границ памяти, техники быстрого запоминания информации и обзор технологий для начала разработки приложения.

Во втором разделе рассмотрены существующие аналоги, определены основные требования к разрабатываемой системе, выбраны метод запоминания и тренировки памяти, определены некоторые алгоритмы запоминания.

В третьем разделе представлено обоснование выбора средств разработки, описана разработка, и проведено статистическое тестирование разработанного алгоритма.

**РАЗДЕЛ 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

В данном разделе будут рассмотрены основные понятия человеческой памяти, ее виды, способы улучшения и расширения границ памяти, техники быстрого запоминания информации и обзор технологий для разработки приложения.

**1.1 Основные понятия человеческой памяти**

Для начала необходимо дать определение человеческой памяти. Память - это когнитивная функция, позволяющая человеку сохранять, хранить и восстанавливать информацию. Она играет важную роль в жизни людей, помогая учиться, принимать решения и функционировать в повседневной жизни[1].

**1.1.1 Компоненты памяти по времени удержания информации**

Память состоит из нескольких основных компонентов, которым необходимо согласованно работать, чтобы вся система человеческой памяти функционировала корректно:

1. Сенсорная память. Это первый этап памяти, в котором происходит прием и сохранение информации через органы чувств. Она имеет ограниченную емкость и длительность хранения.

2. Кратковременная память. Информация из сенсорной памяти, считающаяся важной, переходит в кратковременную память. Эта форма имеет ограниченную емкость и может сохранять информацию от нескольких секунд до минуты.

3. Рабочая память. Представляет собой многокомпонентную систему временного хранения и переработки информации. Она позволяет хранить и использовать информацию в течение короткого времени при выполнении определенных задач или решении проблем[2].

4. Долговременная память. Это длительное хранилище информации, где она может сохраняться в течение многих лет. Долговременная память подразделяется на два основных типа: объявительная, или декларативная, и процедурная память. Декларативная память относится к фактам и событиям, которые человек может сознательно вспоминать, а процедурная – к навыкам и процедурам, которые выполняются автоматически.

**1.1.2 Виды памяти по типу информации**

Виды памяти могут быть также классифицированы по типу информации:

1. Вербальная, или лексическая, память. Она относится к сохранению и воспроизведению слов, языковых выражений и текстов. Данная форма памяти позволяет запоминать и использовать языковые знания, проводить коммуникацию с другими людьми.

2. Визуальная память связана с хранением, а также воспроизведением, визуальных образов, таких как лица, места, предметы и картинки. Визуальная память играет важную роль в распознавании и узнавании окружающей среды.

3. Пространственная память помогает запоминать и извлекать информацию о пространственной организации и расположении объектов в окружающей среде. Пространственная память позволяет ориентироваться в пространстве и проводить навигацию по нему.

4. Процедурная память. Этот вид памяти связан с хранением и воспроизведением навыков, процедур и автоматизированных действий. Данная форма памяти позволяет выполнять различные задачи и действия без необходимости сознательного контроля.

5. Эпизодическая память относится к запоминанию конкретных событий, которые человек пережил в определенное время и в определенном месте. Эпизодическая память включает в себя информацию о контексте, эмоциях и деталях событий[3].

**1.1.3 Классификация по задействованному органу чувств**

Человек осознает мир, опираясь на пять чувств восприятия: зрение, вкус, слух, обоняние и осязание. Человеческая память устроена таким образом, что может основываться на этих чувствах. В зависимости от этого существует следующая классификация:

1. Зрительная память относится к запоминанию визуальной информации, полученной через зрение. Такая память позволяет человеку распознавать образы, цвета и формы.

2. Аудио-вербальная память связана с сохранением и извлечением устной информации, такой как речь, звуки и слова. Данный тип памяти позволяет помнить и воспроизводить слуховые впечатления.

3. Тактильная память относится к хранению тактильной информации, получаемой через касание и осязание. Данная информация подразумевает ощущения текстуры, формы и поверхности объектов. Тактильная память помогает распознавать и запоминать ощущения при взаимодействии с окружающими предметами.

4. Обонятельная память связана с сохранением запахов, получаемых с помощью обоняния. Обонятельная память является сильным фактором в формировании воспоминаний и эмоций. Запахи могут вызывать яркие и длительные воспоминания о прошлых событиях и ситуациях.

5. Вкусовая память относится к запоминанию и воспроизведению вкусовых впечатлений. Вкусовая память связана с запоминанием вкусов и ассоциаций, вызванных определенными вкусами и пищевыми продуктами.

**1.1.4 Фазы памяти**

Память проходит через несколько фаз, которые включают захват информации, ее сохранение, хранение и воспроизведение. Различные фазы памяти имеют свои особенности и процессы.

Первой фазой является захват информации (кодирование). На этом этапе информация впервые поступает в сознание человека. Захват информации включает восприятие и регистрацию сенсорных впечатлений через органы чувств. Во время этой фазы информация обрабатывается и кодируется для последующего сохранения. Например, при прочтении текста, информация визуализируется и кодируется в визуальной форме.

Следующий процесс - сохранение (консолидация). После захвата информации, она проходит процесс консолидации, во время которого информация стабилизируется и сохраняется в памяти. Консолидация запускает процесс передачи информации из кратковременной памяти в долговременную. Этот процесс включает физиологические и химические изменения в мозге, которые укрепляют связи между нейронами и способствуют долгосрочному сохранению информации.

Хранение информации. Эта фаза представляет собой хранение информации в долговременной памяти. Долговременная память является более стабильной и долговечной формой, где информация может храниться в течение длительного времени. Она включает различные типы, такие как объявительная память, другое название – декларативная, и процедурная память, которые были описаны ранее.

В фазе воспроизведения информация извлекается из памяти и используется для воспроизведения или вспоминания. Воспроизведение информации может быть активным или пассивным. Активное воспроизведение включает осознанное вспоминание, когда человек пытается активно восстановить информацию из памяти. Пассивное воспроизведение происходит, когда информация автоматически появляется в сознании без активных усилий.

Фазы памяти взаимосвязаны и взаимодействуют друг с другом. Захваченная информация должна быть успешно закодирована и консолидирована, чтобы быть правильно сохраненной в долговременной памяти. Затем, когда необходимо вспомнить информацию, она должна быть успешно извлечена из памяти[2].

На каждую фазу памяти могут влиять различные факторы: эмоциональное состояние, внимание и мотивация. Эмоционально значимые события могут быть лучше запомнены, чем нейтральные. Кроме того, высокая степень внимания и заинтересованности может повысить эффективность кодирования и консолидации информации.

**1.2 Подходы к улучшению памяти**

Для улучшения памяти и эффективного использования ее фаз можно применять различные стратегии и приемы. Некоторые из них включают повторение и репетицию информации, другие - организацию и структурирование, а также применение мнемотехнических устройств и техник.

Существуют некоторые общие подходы, которые могут помочь улучшить память в целом.

Самый важный подход - здоровый образ жизни. Забота о физическом и психическом здоровье имеет прямое влияние на память. Регулярное физическое упражнение, здоровое питание, достаточный сон и управление стрессом способствуют оптимальной работе мозга и памяти.

Активное участие в ментальных задачах и упражнениях способствует развитию и поддержанию памяти. Это может включать чтение, решение головоломок, участие в интеллектуальных играх, изучение новых навыков или изучение новой информации.

Установка приоритетов и организация информации. Память может быть лучше поддержана, если уделяется достаточное внимание информации, которую человек хочет запомнить. Определение важности и ценности информации поможет сосредоточиться и сделать ее более значимой. Систематическая организация информации помогает улучшить запоминание и воспроизведение. Данный подход обязует использовать методы структурирования, классификации и связей для упорядочивания и связывания информации в логические и смысловые кластеры. Следует избегать отвлекающих факторов и создать благоприятную среду для сосредоточенной работы.

Создание ассоциаций и связей между новой информацией и существующими знаниями помогает укрепить память и облегчить ее вспоминание. В данном методе используют метафоры, аналогии, истории или личные ассоциации, чтобы связать новую информацию с чем-то, что уже знакомо и запоминается лучше.

Для большинства людей повторение является самым эффективным способом закрепления информации в памяти. Повторение информации через определенные промежутки времени (распределение повторений) помогает укрепить запоминание в долговременной памяти.

Для тех людей, кому не подходит метод с частыми повторениями, разнообразие и эмоциональное включение помогают улучшить память. Помогают различные подходы для изучения и запоминания информации, такие как чтение, обсуждение, письменное составление и объяснение другим. Важно создавать эмоциональную связь с информацией, поскольку эмоционально значимые события и факты лучше запоминаются. Можно использовать визуализированную информацию, представляя ее в виде образов или сцен, пользоваться мнемоническими устройствами, такими как акронимы, рифмы или ключевые слова, чтобы помочь запомнить и восстановить нужную информацию[5][6].

**1.3 Техники для запоминания**

Интересная информация запоминается и усваивается лучше, нежели скучная и не представляющая интереса для человека. Это значит, что лишенную ярких черт информацию нужно этими чертами наделить, сделать вызывающей. Для этого было придумано множество техник для запоминания, которые затрагивают разные аспекты и виды памяти. Эти методы не только позволяют с большей скоростью и эффективностью запомнить информацию, но и тренируют память в целом. Основная задача человека – подобрать ту технику, которая будет работать именно для него[5].

**1.3.1 Метод мнемоники**

Метод мнемоники - это техника запоминания, основанная на использовании ассоциаций, образов и структурирования информации для более эффективного запоминания и восстановления.

Основная идея метода мнемоники заключается в том, чтобы превратить абстрактную или сложную информацию в образы, ассоциации или структуры, которые запоминаются легче. Это делается путем создания связей между новой информацией и уже известными, легко запоминающимися элементами. Такие связи помогают укрепить запоминание и упростить восстановление информации.

В мнемонике существует множество техник, которые помогают улучшить запоминание и облегчить восстановление информации в памяти. Вот несколько распространенных техник, которые широко применяются:

1. Метод ключевых слов. Эта техника особенно полезна при запоминании новых терминов. Она основана на создании ассоциаций между новыми словами и уже известными образами. Выбирается ключевое слово, которое звучит похоже на новое, и создается визуальная ассоциация между ними. Это помогает запомнить новый термин и легче его восстановить.

2. Метод ассоциаций. Этот метод основан на создании ассоциаций или связей между элементами, которые нужно запомнить. Выбирается главное слово или понятие, которое необходимо сохранить в памяти, и создается цепочка ассоциаций, связывающих его с другими словами или образами. Чем более яркие, необычные и эмоционально насыщенные ассоциации, тем легче будет запомнить информацию.

3. Метод акронимов. В этом методе используются первые буквы или слоги каждого элемента, которые нужно запомнить, для создания нового слова или фразы. Это помогает легче запоминать длинные списки или последовательности.

4. Метод цепочки. Этот метод основан на создании связных историй, где элементы, которые нужно сохранить, становятся частью повествования. Слово тянет за собой цепочку из других слов, которые связываются между собой, формируя полноценную историю.

5. Метод рифм (рифмизация и акростих). В этом методе используются рифмы и звуковые ассоциации для запоминания информации. Необходимо создать фразы или стихи, где каждый элемент, который нужно запомнить, рифмуется или звучит похоже на предыдущий или следующий элемент. Это помогает создать музыкальную и легко запоминающуюся структуру.

Каждая из этих техник имеет свои преимущества и может быть эффективной в различных ситуациях. Важно экспериментировать и выбрать ту, которая лучше всего соответствует предпочтениям и типу информации, которую нужно запомнить.

Преимущества метода мнемоники:

1. Улучшение запоминания. Создание связей и образов помогает сделать информацию более живой и легко запоминающейся.

2. Улучшение организации. Мнемоника помогает структурировать информацию, делая ее более организованной и легкой для запоминания.

3. Большая гибкость. Метод мнемоники довольно гибок и может быть применен к различным видам информации.

4. Персонализация. Мнемоника позволяет каждому человеку создавать свои собственные ассоциации и образы.

Недостатки метода мнемоники:

1. Ограничения при работе с абстрактными концепциями. Мнемоника может быть менее эффективной при работе с абстрактными или сложными концепциями, которые не имеют четких ассоциаций или образов. В таких случаях мнемоника может потребовать дополнительных усилий и творческого подхода для создания эффективных ассоциаций.

2. Индивидуальные предпочтения и предрасположенности: Некоторым людям может быть сложно создавать и использовать образы и ассоциации в своей памяти. Каждый человек имеет свои индивидуальные предпочтения и способы запоминания, поэтому метод мнемоники может не подходить всем пользователям.

3. Ограниченная применимость: Метод мнемоники может быть наиболее полезным для запоминания конкретных фактов, списков или последовательностей. Он может быть не применим для более сложных структур, концепций или абстрактных идей, которые требуют глубокого понимания и анализа.

Метод мнемоники позволяет создавать ассоциации, образы и структуры, которые облегчают запоминание и восприятие информации. Однако, для достижения оптимальных результатов, необходимо учитывать индивидуальные предпочтения и особенности каждого человека, а также подбирать подходящие ассоциации и образы для конкретного типа информации.

**1.3.2 Метод локусов**

Метод локусов, также известный как метод помещений или метод чертогов разума, имеет древние корни и был использован в течение многих веков для запоминания информации. Его истоки связываются с древнегреческими и римскими ораторами, которые использовали эту технику для запоминания и представления длинных многочасовых речей перед аудиторией.

Одним из наиболее известных приверженцев метода локусов был древнегреческий поэт Симонид из Кеоса, который жил в V веке до н.э. Во время одного из своих выступлений в храме, потолок обвалился, и Симонид был способен точно вспомнить имена и расположение каждого присутствовавшего в зале. Из этого случая он вывел принцип использования визуализации и пространственной организации для запоминания информации.

Метод локусов стал широко известен в средние века благодаря французскому монаху Доминикусу Виньи, который использовал эту технику для запоминания и восстановления библейских текстов.

Основная идея метода локусов заключается в том, чтобы связать каждый элемент информации с определенным местом или локацией в воображаемом пространстве, таком как дом, офис или любое другое место, которое хорошо знакомо человеку. Необходимо представлять себя, проходящего по этим локациям и визуализировать информацию, связанную с каждым местом.

Подробный процесс использования метода выглядит следующим образом:

1. Выбор воображаемого места. Это должно быть место, которое можно визуализировать в деталях. Это может быть дом, офис, школа или любое другое место, где часто бывает человек.

2. Определение локаций. Выбранное место делится на определенное количество локаций или мест, например, комнаты, мебель или другие ярко выраженные объекты. Они должны быть легко представляемы и между этими локациями необходимо свободное перемещение.

3.Связь информации с локациями. Для каждого элемента информации, который необходимо запомнить, создается яркая и запоминающаяся визуализация, связанная с конкретной локацией. Чем более необычная визуализация, тем лучше.

4. Процесс визуализации. Необходимо представить себя, проходящего по установленным ранее локациям и видящего каждую связанную с ними визуализацию. Нужно стараться визуализировать детали и включать все чувства: звуки, запахи, текстуры и т. д.

5. Перемещение по локациям. Перемещение должно происходить последовательно по локациям. Визуализируется каждая связанная с ними информация. Во время восстановления информации можно снова визуализировать процесс движения и вспоминать связанную с каждой локацией информацию.

Метод локусов основывается на принципе ассоциации и визуализации, которые являются мощными инструментами для запоминания информации. Чтобы использовать метод локусов более эффективно необходимо создавать яркие и живые образы, использовать чувственные впечатления. Визуализация должна быть как можно более реалистичной и детальной, подключать разные чувства, помимо зрительных образов. Чем больше видов памяти будет затронуто, тем эффективнее станет запоминание с помощью данного метода.

К достоинствам данного метода можно отнести:

1. Запоминание больших объемов информации
2. Гибкость и персонализация
3. Глубокое закрепление информации

Метод имеет следующие недостатки:

1. Требуется практика и время
2. Индивидуальная предрасположенность
3. Ограничения в использовании

Запоминание с помощью техники локусов требует практики и терпения, но постепенно станет возможно запоминать и восстанавливать большие объемы информации.

**1.3.3 Метод сторителлинга**

Метод сторителлинга, или метод использования историй, является эффективным способом запоминания и передачи информации с помощью создания и рассказа историй. Он имеет древние корни и широко применяется в различных областях знаний.

Истории всегда были важным средством передачи информации и опыта от одного поколения к другому. Рассказывая и слушая истории, люди легче запоминают информацию, воспринимают ее более эмоционально и связывают события и факты в единое целое. Метод сторителлинга базируется на этой способности человеческого мозга лучше запоминать и понимать информацию через сюжеты и истории.

Принцип работы метода заключается в создании увлекательных и запоминающихся историй, которые включают в себя элементы информации, которые нужно запомнить. Это может быть последовательность событий, характеры, места и другие детали, которые помогают воссоздать контекст и связь между элементами.

При использовании метода сторителлинга важно учесть следующие аспекты:

1. Эмоциональная привлекательность. Истории, которые вызывают эмоциональный отклик, легче запоминаются. Поэтому важно использовать элементы драмы, интриги или эмоций в создании истории.

2. Визуализация. Чем более ярко человек представляет события, персонажей и детали истории, тем легче они будут запоминаться. Визуализация помогает создать образы в памяти, которые легко восстанавливаются.

3. Связь с информацией. Важно, чтобы элементы истории были связаны с информацией, которую необходимо запомнить. Это может быть с помощью метафор, аналогий или прямых ассоциаций.

4. Повторение. Повторение истории помогает закрепить информацию в памяти. Можно использовать повторения в разных контекстах или вариациях истории, чтобы укрепить запоминание.

Метод сторителлинга в современном мире продолжает развиваться и применяться в различных сферах. Он позволяет создать эмоциональную связь между человеком и информацией, делая процесс запоминания и передачи знаний более увлекательным и эффективным.

Плюсы метода:

1. Запоминаемость. Человеческий мозг лучше запоминает и связывает информацию, когда она представлена в виде истории с определенным сюжетом и эмоциональной привлекательностью.

2. Понимание и связывание. Истории помогают людям создать контекст и структуру для новой информации, что облегчает ее усвоение и применение.

3. Эмоциональная привлекательность. Использование метода сторителлинга позволяет создавать эмоциональную связь с информацией.

Минусы сторителлинга:

1. Время и подготовка. Создание качественных историй требует времени и тщательной подготовки. Это может потребовать дополнительных усилий и ресурсов для их разработки и рассказа.

2. Адаптация. Не всегда легко адаптировать истории под нужный контекст.

3. Отсутствие точности. Истории могут быть более эмоциональными и артистичными, но при этом некоторые детали могут быть искажены или упрощены.

4. Ограничение использования. Метод сторителлинга может быть не подходящим для всех видов информации или контекстов. В некоторых случаях, особенно в научных или технических областях, более прямой и точный подход может быть предпочтительным.

5. Ограниченность объемом информации: Вместимость истории ограничена по объему, что может быть препятствием при передаче большого количества информации или сложных концепций. В этом случае может потребоваться комбинирование с другими методами или подходами.

Важно учитывать, что плюсы и минусы метода могут зависеть от конкретной ситуации и целей использования.

**1.3.4 Метод интервального повторения**

Метод интервального повторения является эффективным подходом к тренировке памяти и усвоению информации. Он основан на принципе систематического повторения материала с определенными временными интервалами. Главная идея метода заключается в том, чтобы повторять информацию в определенные моменты времени, исходя из степени ее усвоения и забывания.

Принцип работы метода интервального повторения основан на знаменитой "кривой забывания", сформулированной Германом Эббингаузом. Согласно этой кривой, новая информация начинает забываться очень быстро после изучения, но затем скорость забывания уменьшается, и информация сохраняется в памяти на более длительное время.

Применение метода интервального повторения позволяет оптимально использовать эту кривую забывания. Информация повторяется в определенные моменты времени, когда вероятность забывания становится выше. Таким образом, повторение происходит в самый оптимальный момент для закрепления знаний.

Существует несколько формул и методов для определения правильного интервала повторения в методе интервального повторения: метод Pimsleur (основан на работе Пола Пимслера), метод Зинкера (формула не фиксирована и может меняться), метод Лукретии (использует принцип плавающих интервалов).

Преимущества метода интервального повторения:

1. Эффективность. Метод позволяет оптимизировать процесс запоминания и усвоения информации, обеспечивая эффективное запоминание на длительный срок.

2. Экономия времени. Благодаря оптимальному использованию времени повторений, метод интервального повторения позволяет сэкономить время и энергию, поскольку более значимые и сложные материалы повторяются чаще, а менее значимые реже.

3. Индивидуальный подход. Метод интервального повторения адаптируется к индивидуальным потребностям и способностям каждого человека. Интервалы повторений могут быть настроены в соответствии с индивидуальными особенностями и склонностями к забыванию.

4. Усиление долговременной памяти. Повторение информации в определенные временные интервалы способствует закреплению знаний в долговременной памяти.

5. Заполнение пробелов. Метод интервального повторения позволяет выявить пробелы в знаниях и активно работать над их заполнением.

6. Поддержка мотивации. Поскольку метод интервального повторения предполагает постепенное увеличение интервалов между повторениями, он может способствовать укреплению мотивации и чувству прогресса.

Минусы метода интервального повторения:

1. Индивидуальная настройка. Некоторым людям может потребоваться больше или меньше повторений, чтобы закрепить знания, поэтому необходимо найти оптимальные интервалы для каждого человека.

2. Сложность оценки интервалов. Определение оптимальных интервалов повторения может быть сложной задачей.

3. Отсутствие креативности. В отличие от некоторых других методов запоминания, метод интервального повторения не стимулирует креативное мышление или ассоциативное мышление. Он сконцентрирован на повторении и восстановлении информации, нежели на создании связей или новых ассоциаций.

Важно отметить, что эффективность метода интервального повторения может варьироваться в зависимости от индивидуальных особенностей и предпочтений каждого человека.

**1.3.5 Чанкинг-метод**

Метод дробления (chunking) является одной из техник, используемых для улучшения запоминания информации. Этот метод основан на принципе разделения большого объема информации на более мелкие и управляемые части, называемые "дроблениями". Каждый кусок представляет собой более маленькую и понятную единицу информации, которую легче запомнить и обработать.

Истоки метода дробления связаны с исследованиями памяти и когнитивной психологии. Одним из ранних исследователей в этой области был Джордж Миллер, который в 1956 году опубликовал статью "Волшебное число семь плюс-минус два: некоторые ограничения наших способностей к обработке информации". В этой статье он указал на то, что наша рабочая память способна эффективно обрабатывать около семи элементов информации одновременно.

Принцип работы метода дробления основывается на ограничениях рабочей памяти и ее способности к обработке ограниченного количества информации. Путем разбиения большого объема информации на более мелкие куски мы можем увеличить вероятность успешного запоминания и обработки этой информации.

Олицетворением данного метода является дробление текста на параграфы и абзацы. Также данный метод находит отражение в записи мобильных номеров. Для человеческого восприятия ощутима разница между записью 88005553535 и 8 (800) - 555- 3535.

Применение метода дробления помогает справиться с перегрузкой информацией, разбивая ее на более управляемые и понятные куски. Это позволяет более эффективно использовать рабочую память, улучшает запоминание и обработку информации, а также упрощает выполнение сложных задач.

Метод фрагментации имеет свои плюсы и минусы, которые следует учитывать при его использовании.

Плюсы метода дробления:

1. Улучшение запоминания. Разбиение информации на более мелкие куски позволяет лучше удерживать ее в памяти из-за ограничений рабочей памяти.

2. Улучшение организации информации. Метод дробления помогает организовать информацию в более структурированном и логическом виде.

3. Снижение перегрузки информацией. Разделение большого объема информации на более мелкие куски упрощает ее обработку и уменьшает перегрузку информацией.

4. Применимость к различным областям. Метод может быть применен как для запоминания текстовой информации, так и для числовых данных, последовательностей и других типов информации.

Минусы метода дробления:

1. Потребность в дополнительной обработке. Разбиение информации на более мелкие куски может занимать время и требовать дополнительных шагов для разделения и организации информации.

2. Индивидуальные предпочтения и стиль обучения. Эффективность метода дробления может зависеть от индивидуальных предпочтений и стиля обучения каждого человека.

3. Ограничение в применимости: В некоторых случаях, таких как изучение абстрактных концепций или сложных теорий, более глубокое понимание и анализ может оказаться эффективнее, чем разделение на мелкие части.

4. Зависимость от контекста: Разбиение информации на мелкие куски может привести к потере контекста, особенно если связи между этими кусками не являются явными или логическими.

Эффективность метода дробления может варьироваться в зависимости от конкретной ситуации, типа информации, индивидуальных предпочтений и стилей обучения. Некоторым людям может подойти разбиение информации на мелкие куски и их последовательное запоминание, в то время как другие могут предпочитать сохранять информацию в более цельном и связанном виде.

**Вывод**

В данном разделе было раскрыто понятие человеческой памяти и исследованы ее виды. В ходе анализа были рассмотрены различные методы и техники для запоминания и тренировки памяти. Изучены различные подходы, которые помогают улучшить процесс запоминания информации и повысить эффективность обучения. Каждый из этих методов имеет свои особенности и применяется в различных ситуациях.

Методы и техники, такие как метод локусов, мнемоника, сторителлинг, интервальное повторение и метод дробления, предоставляют разнообразные инструменты для работы с информацией и ее запоминания. Они основаны на принципах визуализации, ассоциации, контекста, повторения и структурирования, которые способствуют более глубокому усвоению и долговременному сохранению информации.

Каждый из этих подходов имеет свои плюсы и минусы. Некоторые методы могут быть более подходящими для конкретных типов информации или стилей обучения. Некоторые могут требовать больше времени и усилий для освоения, в то время как другие могут быть более простыми и быстрыми в использовании.

**РАЗДЕЛ 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ**

В данном разделе будут рассмотрены существующие аналоги, определены основные требования к разрабатываемой системе, выбраны метод запоминания и тренировки памяти, определены некоторые алгоритмы запоминания.



**2.1 Изучение существующих реализаций**

Существует множество приложений для запоминания, которые предоставляют различные функциональные возможности. Кто-то найдет среди них подходящее под собственные нужды. Однако для российского пользователя большинство приложений окажутся не доступными, так как, являясь разработанными иностранными компаниями, они не поддерживают русский язык.

**2.1.1 Flashcards Maker**

Flashcards Maker - это приложение для создания и изучения карточек для запоминания информации. Позволяет пользователям легко создавать и настраивать карточки с помощью текста, изображений и звуковых файлов. Это мощное приложение, которое помогает улучшить память и обучение.

Flashcards Maker имеет ряд преимуществ, но также имеет некоторые недостатки. Рассмотрим их подробнее.

Плюсы приложения Flashcards Maker:

1. Простота использования. Приложение предлагает интуитивно понятный интерфейс и удобные инструменты для создания и настройки карточек. Это делает процесс работы с приложением легким даже для новичков.

2. Гибкость и настройка. Flashcards Maker предоставляет широкие возможности для настройки карточек и повторений. Позволяет настроить интервалы повторений, добавлять изображения и звуковые файлы, организовывать карточки по категориям и многое другое.

3. Интерактивность и эффективность. Приложение предлагает интерактивные возможности, такие как звуковые эффекты и анимация, которые способствуют более эффективному запоминанию информации. Это делает процесс обучения более интересным и привлекательным.

4. Отслеживание прогресса. Приложение предоставляет отчеты и статистику о прогрессе в изучении и запоминании информации. Это позволяет оценить достижения и определить области, требующие дополнительной тренировки.

Недостатки приложения Flashcards Maker:

1. Отсутствие совместного использования. Flashcards Maker не всегда предлагает возможность совместного использования карточек с другими пользователями. Если появляется необходимость обмениваться карточками или работать в команде, это может быть ограничением.

2. Ограничения бесплатной версии. Некоторые функции или возможности могут быть доступны только в платной версии приложения. Бесплатная версия может иметь некоторые ограничения или наличие рекламы, что может быть неудобством для пользователей.

Flashcards Maker является крайне полезным и эффективным приложением для создания и изучения карточек. Оно предлагает множество функций, которые помогают улучшить запоминание информации и облегчить процесс обучения. Однако перед использованием приложения стоит учесть его ограничения и соответствие пользовательских потребностей и предпочтений.

**2.1.2 AnkiDroid**

AnkiDroid - это приложение для запоминания и повторения информации с использованием методики интервальных повторений. Оно предоставляет пользователю гибкий и эффективный способ организации и повторения знаний, позволяя улучшить запоминание и удержание информации на долгосрочной основе.

Плюсы AnkiDroid:

1. Мощная методика интервальных повторений, которая помогает улучшить запоминание и удержание информации.
2. Гибкие настройки повторений, позволяющие адаптировать процесс обучения под потребности пользователя.
3. Возможность синхронизации и обмена карточками между разными устройствами.
4. Богатый выбор настроек, тем оформления и расширений для персонализации приложения.

Минусы AnkiDroid:

1. Несмотря на мощные возможности, AnkiDroid может показаться сложным в использовании для новичков. Он требует времени и усилий для изучения его функциональности и настройки.
2. Некоторые пользователи могут почувствовать ограничение в дизайне и пользовательском интерфейсе приложения, которое может быть немного устаревшим и менее интуитивным.
3. Приложение может потреблять больше ресурсов и занимать место на устройстве из-за большого объема карточек и данных.

AnkiDroid предоставляет множество функций и возможностей для эффективного запоминания, персонализации и отслеживания прогресса. Однако его сложность использования и ограничения в дизайне могут быть минусами для некоторых пользователей.

**2.1.3 MemoWorld**

Приложение "MemoWorld" является мобильным приложением для запоминания и тренировки памяти, предназначенным для изучения иностранного языка с помощью карточек.

Плюсы "MemoWorld":

1. Гибкие настройки и возможность персонализации.
2. Применение методики интервальных повторений для оптимизации запоминания.
3. Разнообразные режимы тренировки.
4. Отслеживание прогресса и результатов обучения.

Минусы "MemoWorld":

1. Ограниченный выбор предустановленных карточек и колод.
2. Возможны некоторые ограничения в бесплатной версии приложения.
3. Некоторые пользователи могут ощущать необходимость в дополнительных функциях или интеграции с другими платформами.
4. Может показаться сложным в использовании для новичков.
5. Основное назначение – обучение языкам.

" MemoWorld " предлагает возможности для изучения и запоминания информации в области изучения языков. Однако запоминание иной информации с помощью данного приложения реализуется слабо.

**2.1.4 Memrise**

Приложение "Memrise" - это мобильное приложение для изучения и запоминания языков. Оно предлагает уникальные методы и техники, разработанные для эффективного запоминания информации.

Плюсы "Memrise":

1. Использование мнемонических техник для более эффективного запоминания.
2. Интерактивные тренировки, которые делают процесс обучения интересным и вовлекающим.
3. Адаптивное повторение, учитывающее прогресс и потребности.

Минусы "Memrise":

1. Некоторые функции и возможности могут быть доступны только в платной версии приложения.
2. Ограниченность содержания для некоторых языков и предметов.
3. Иногда возникают технические проблемы или задержки в работе приложения.
4. Основное направление приложения – обучение языкам.

"Memrise" предлагает широкие возможности для изучения и запоминания информации в области изучения языков. Однако запоминание иной информации с помощью данного приложения не поддерживается.

**2.2 Определение требований к системе**

Определение требований к разрабатываемой системе – важный этап процесса разработки. Данный этап позволяет установить основные цели и функциональные возможности приложения, которые должны быть учтены в процессе его разработки.

1. Функциональные требования:

1.1 Добавление и редактирование материала. Пользователям должна быть предоставлена возможность добавлять и редактировать контент, такой как карточки с вопросами и ответами.

1.2 Организация материала. Приложение должно обеспечивать удобную организацию контента, например, с помощью создания коллекций и добавления цвета.

1.3 Управление временем и интервалами. Приложение должно устанавливать временные интервалы для повторения материала и эффективного управления тренировкой памяти.

1.4 Синхронизация и доступность. Приложение должно обеспечивать синхронизацию данных между устройствами пользователя.

2. Интерфейс и пользовательский опыт:

2.1 Привлекательный и интуитивно понятный интерфейс. Приложение должно иметь привлекательный дизайн и понятную навигацию, чтобы пользователи могли легко освоить его использование.

2.2 Поддержка различных типов контента: Приложение должно поддерживать разнообразные типы контента, такие как текст, изображения и звуковые файлы, для упрощения процесса запоминания.

2.3 Регулирование уровня запоминания. Пользователям должна быть предоставлена возможность оценить уровень запоминания материала в соответствии с их потребностями и способностями.

2.4 Лояльность к мобильным устройствам. Приложение должно быть оптимизировано для мобильных устройств, обеспечивая удобное и эффективное использование на различных экранах и сенсорных интерфейсах.

3. Безопасность и конфиденциальность:

3.1 Защита пользовательских данных. Приложение должно обеспечивать надежную защиту пользовательских данных, включая персональную информацию и прогресс тренировки.

3.2 Аутентификация и авторизация. Приложение может предоставлять механизмы аутентификации и авторизации пользователей, например, с помощью аутентификации через аккаунт Google.

**2.3 Выбор используемых технологий и языка программирования**

Для разработки мобильного приложения для запоминания и тренировки памяти необходимо выбрать подходящие технологии и язык программирования.

1. Язык программирования. Kotlin - это официальный язык разработки для Android, который обладает высокой производительностью, безопасностью и поддержкой современных концепций. Kotlin обладает интуитивным синтаксисом и позволяет использовать множество библиотек и инструментов из экосистемы Android. Кроме того, Kotlin предлагает набор инструментов, которые помогают предотвратить распространенные ошибки программирования, такие как нулевые указатели. Он имеет строгую систему типов и поддерживает функции безопасного вызова, что способствует созданию надежного приложения.

2. Фреймворк разработки мобильных приложений. Android SDK - это набор инструментов и библиотек, предоставляемых Google для разработки приложений под платформу Android. Он содержит все необходимые компоненты для создания пользовательского интерфейса, взаимодействия с устройством и работой с базами данных.

3. Интегрированная среда разработки. Android Studio - это официальная интегрированная среда разработки для создания Android-приложений. Она предоставляет широкий набор инструментов и функциональных возможностей, таких как автозаполнение кода, отладка и эмуляторы устройств. Android Studio основана на среде разработки IntelliJ IDEA и предлагает удобную среду для разработки на языке Kotlin.

4. База данных. Firebase - это облачная платформа, предоставляемая компанией Google. Она предлагает набор инструментов и сервисов, которые помогают разработчикам создавать высококачественные приложения, упрощая задачи, такие как хранение данных, аутентификация пользователей, управление уведомлениями и другие. Firebase поддерживает хранение и синхронизацию данных внутри приложения между клиентами в режиме реального времени.

Выбранные технологии и инструменты предоставляют широкие возможности для разработки мобильного приложения для запоминания и тренировки памяти на платформе Android. Они обеспечивают эффективность, удобство разработки и хорошую производительность приложения.

**2.4 Определение алгоритма запоминания**

Для запоминания информации существует множество различных методов (раздел 1.3). В разрабатываемом приложении для запоминания и тренировки памяти необходимо совместить несколько подходов, чтобы охватить группы людей, которым подходят разные методы запоминания информации. Помимо визуализации и структуризации материала необходимо реализовать метод интервального повторения. Основная задача – выбор алгоритма для реализации данного метода.

Рассмотрим один из наиболее распространенных алгоритмов - SuperMemo, который представлен следующей формулой (2.1):

I = I \* EF (2.1)

где

I (интервал) - это время, прошедшее с предыдущим повторением,

EF (фактор эффективности) - это числовое значение, которое определяет, насколько успешно пользователь запомнил материал на предыдущем повторении.

Фактор эффективности может быть представлен в виде десятичной дроби. Обычно для нового материала EF устанавливается на значение 2,5. Для каждого последующего повторения вычисляется новый интервал, умножая предыдущий интервал на EF. Если предыдущий интервал был 5 дней, а фактор эффективности равен 2,5, то новый интервал будет равен 5 \* 2,5 = 12,5 дней, что можно округлить до 13 дней.

Эта формула позволяет увеличивать интервалы повторения с каждым успешным запоминанием и адаптироваться к индивидуальным особенностям памяти и скорости забывания. Чем лучше запоминается материал, тем больше увеличивается интервал между повторениями.

Реализация внутри приложения выглядит следующим образом. Пользователю предоставлена карточка, под которой расположены кнопки с цифрами от одного до пяти. Пользователю необходимо оценить степень запоминания карточки. В зависимости от выбранной оценки запоминания будет рассчитана следующая дата повторения показанной карточки.

Далее представлена схема реализованного алгоритма на рисунке 2.4.1.

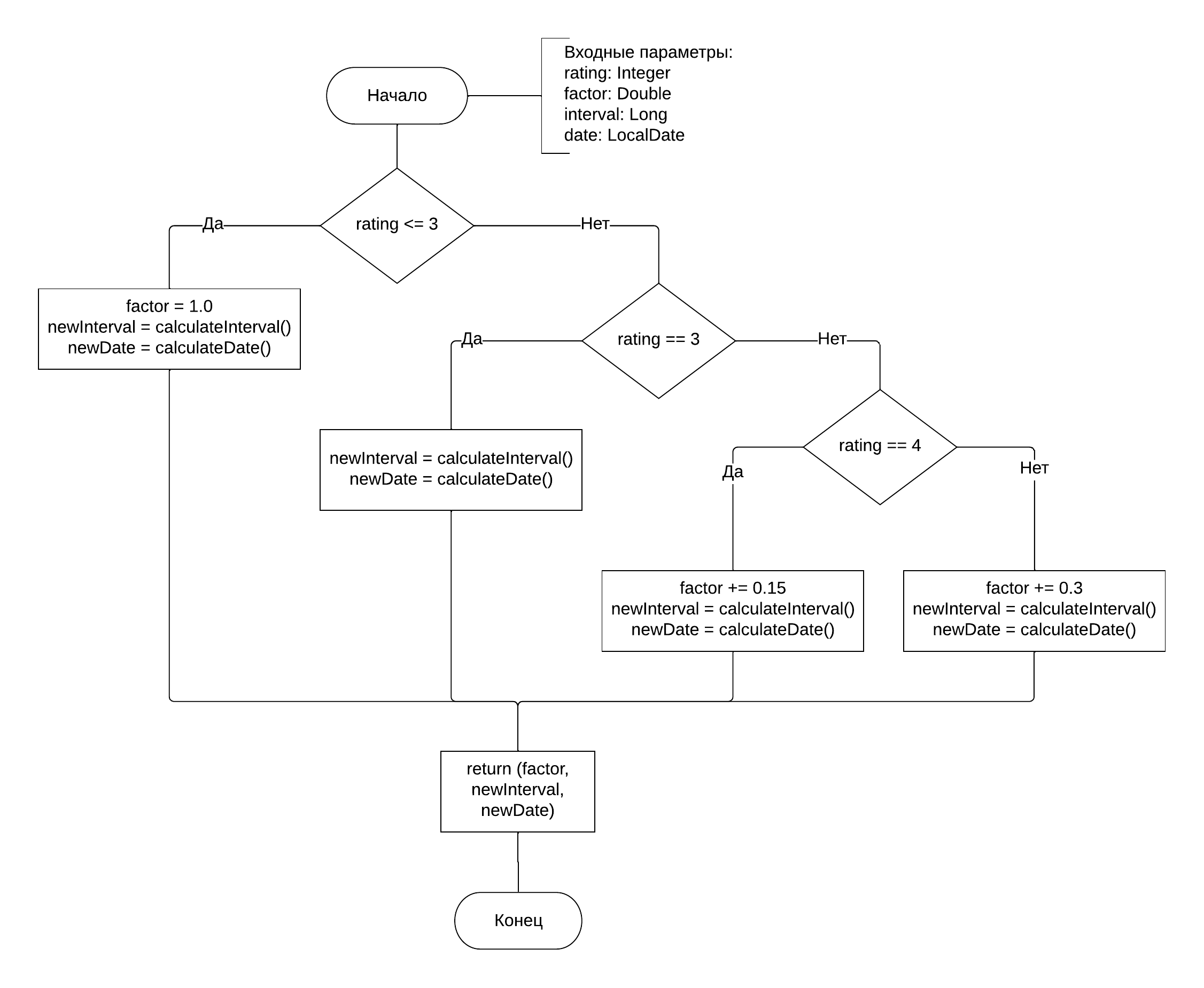


Рисунок 2.4.1 – Схема разработанного алгоритма

На рисунке 2.4.2 представлена блок-схема метода calculateInterval()

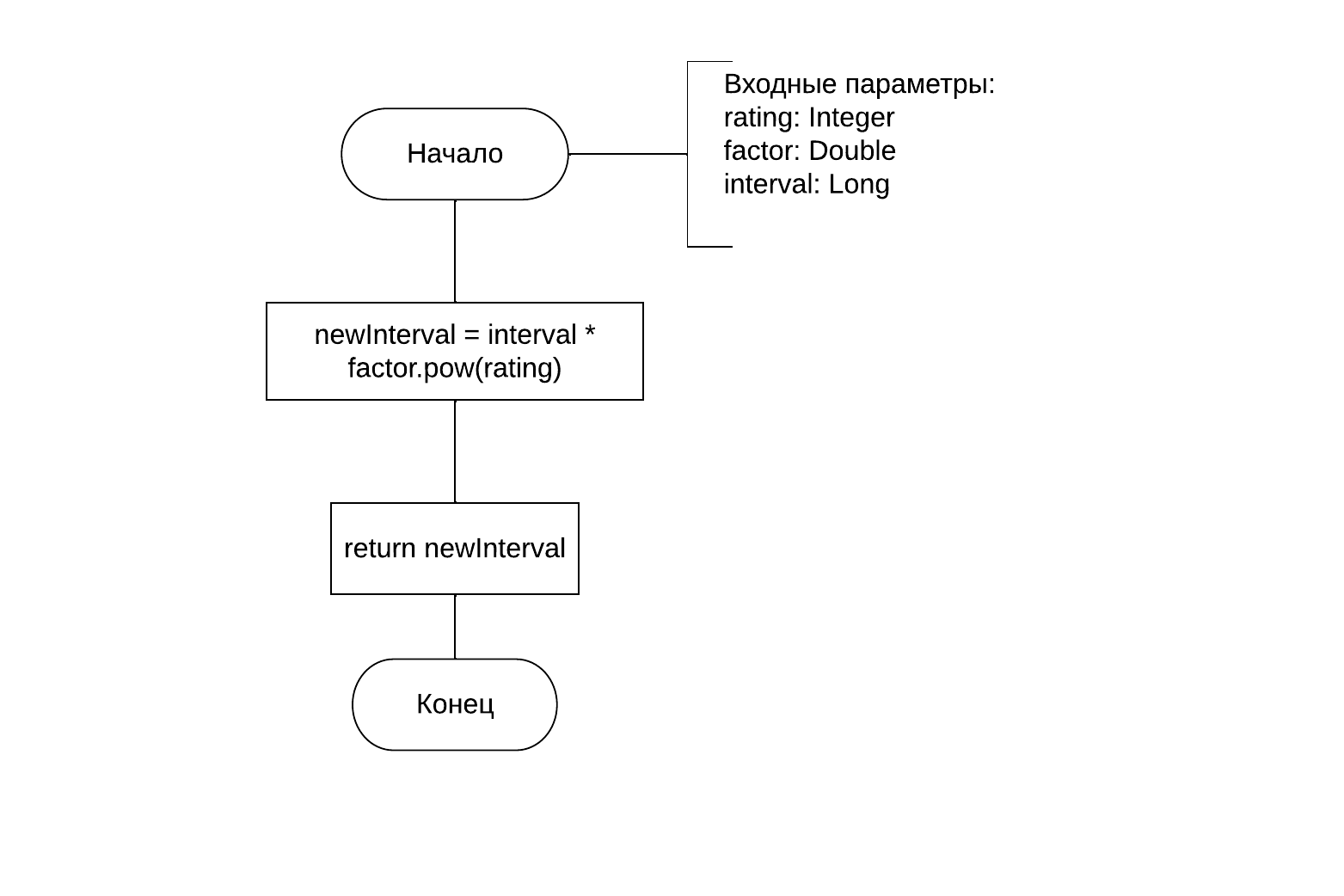


Рисунок 2.4.2 – Схема метода calculateInterval()

На рисунке 2.4.3 представлена блок-схема метода calculateDate():

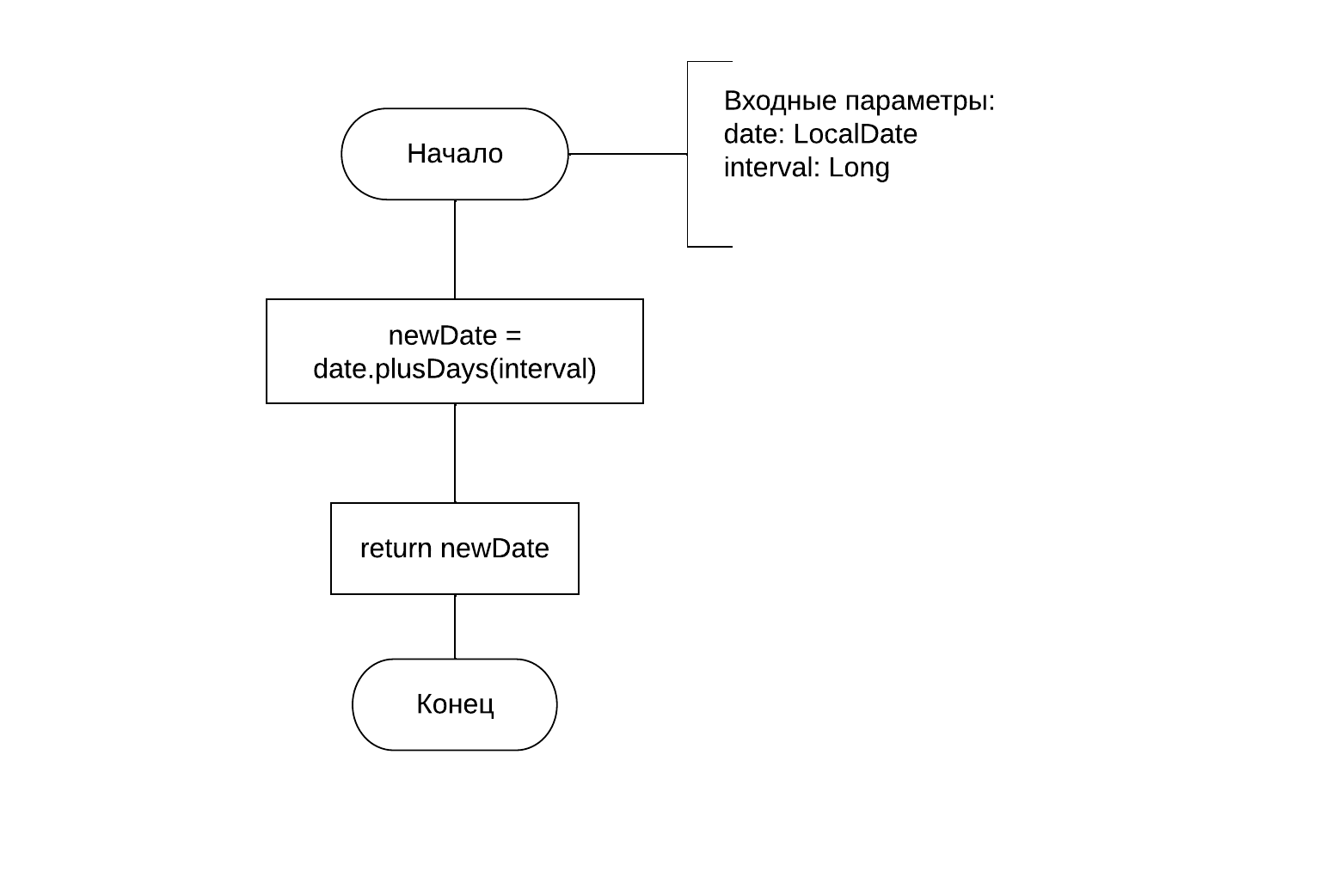


Рисунок 2.4.3 – Схема метода calculateDate()

**2.5 Функциональность приложения**

В ходе проектирования приложения для запоминания и тренировки памяти были выделены следующие классы, необходимые для решения поставленных задач:

LoginActivity – осуществляет аутентификацию пользователя при первом вхождении в приложение. Используется Google аккаунт для входа.

CardAdapter – осуществляет обновление информации в RecuclerView, где отображаются карточки.

CollectionAdapter – осуществляет обновление информации в RecuclerView, где отображаются коллекции.

CardActivity – обрабатывает действия с каждой отдельной карточкой.

CollectionActivity – обрабатывает действия с каждой отдельной коллекцией.

EditCardActivity – класс, с помощью которого осуществляется добавление новой карточки или обновление информации в уже существующей.

EditCollectionActivity – класс, с помощью которого осуществляется добавление новой коллекции или обновление информации в уже существующей.

FirebaseHelper – связывает базу данных и приложение. Содержит методы, которые позволяют правильно обрабатывать содержимое базы данных – получать данные, добавлять, обновлять и удалять.

PlayCardActivity – класс, с помощью которого реализована игра «Найди пару» для тренировки памяти.

RememberCard – отвечает за реализацию метода интервального повторения на основе алгоритма SuoerMemo.

ShulteActivity – класс, с помощью которого реализовано упражнение для тренировки «таблица Шульте».

**Вывод**

В ходе данного раздела были приведены примеры уже существующих реализаций и выделены их достоинства и недостатки. Затем были установлены требования к приложению для запоминания и тренировки памяти и общие функциональные требования к системе, выбран алгоритм запоминания и спроектирован на основе метода SuperMemo. Так же были выделены классы, необходимые для решения поставленной задачи.

**РАЗДЕЛ 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

В данном разделе будет представлена реализация приложения для запоминания и тренировки памяти, работа с базой данных и описание существующих методов. Так же будет приведена работа приложения в ходе тестирования.



**3.1 Создание макета**

Перед началом разработки мобильного приложения для запоминания и тренировки памяти необходимо создать макет с продуманным дизайном и удачными цветовыми решениями. На данном этапе происходит создание удобного и интуитивно понятного пользователю интерфейса, который должен обеспечивать оптимальное использование приложения.

Одним из важных этапов является верный подбор цветовой гаммы интерфейса. Опираясь на показания ученых и психологов, палитра цветов оказывает влияние на усвоение материала и обучаемость человека. Одни цвета стимулируют креативность и воображение, другие – концентрацию и интеллект.

Профессора университета Южной Флориды выяснили, что лучше запоминается информация, представленная с помощью сочетающихся оттенков, нежели с помощью контрастных и ярких, которые рассеивают внимание. Таким образом, для улучшения усвоения информации лучше выбрать разные оттенки одного цвета или гармонирующие между собой оттенки разных неброских цветов.

Опираясь на представленный анализ, основной цветовой палитрой приложения были выбраны оттенки пастельного синего цвета. Множество научных исследований доказывают, что люди, работающие с синим и неброским голубым цветом, являются наиболее продуктивными. В качестве дополняющего цвета решено использовать оранжевый цвет, так как он улучшает функционирование нервной системы и в совокупности с синей цветовой гаммой создаст необходимые условия для высокой концентрации внимания.

Вместе с утверждением цветового решения была разработана интуитивно понятная навигация внутри приложения, позволяющая пользователю не запутаться в обилии настроек перед началом эксплуатации.

На рисунке 3.1.1 представлен макет, иллюстрирующий путь пользователя внутри приложения. Учтены и исправлены все недостатки, выявленные на этапе анализа аналоговых приложений.

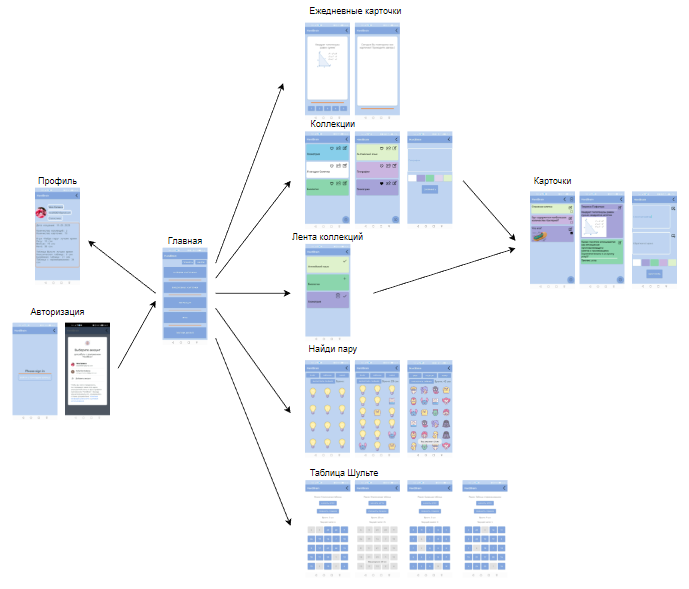


Рисунок 3.1.1 – Макет приложения

Путь пользователя начинается с первого входа в приложение: регистрации и последующей авторизации. Так пользователь попадает на главную страницу, которая является навигатором и содержит элементы, позволяющие перейти на другие активности приложения.

На макете представлен профиль, где пользователь может увидеть личные данные и статистику по механикам приложения.

Приложение имеет список коллекций, в которых находят карточки. При нажатии на иконку «сердечко» коллекция выбирается для проверки знаний в виде ежедневных карточек. При нажатии на иконку редактирования можно изменить название коллекции и ее цвет. При нажатии на иконку «Поделиться» пользователь добавляет созданную коллекцию в общий список коллекций с карточками. При нажатии на коллекцию открывается список карточек, принадлежащий данной коллекции. При длительном нажатии появляются чекбоксы для выбора коллекций на удаление и, соответственно, кнопка удаления. Далее представлен экран редактирования, или создания, карточек. Необходимо указать лицевую и обратную сторону, при необходимости добавить изображения и выбрать цвет карточки.

В приложении содержится активность, где хранятся коллекции с карточками, которыми поделились пользователи. Каждый пользователь может добавить коллекцию из представленной активности, а также удалить коллекцию, если является ее создателем.

На изображении представлен список карточек из определенной коллекции. При нажатии на карточку, она раскрывается и становится видна не только лицевая сторона, но и обратная. Редактирование и удаление происходит аналогично со списком коллекций.

Мобильное приложение содержит активность и ежедневными карточками выбранных коллекций. Необходимо оценить степень запоминания карточки от одного до пяти, чтобы изменить следующую дату повторения карточки. При нажатии на карточку появляется обратная сторона вместо лицевой, и наоборот. При повторении всех карточек появляется соответствующая надпись и кнопки оценки исчезают.

Помимо механик для запоминания информации имеется окно с интерактивной игрой для тренировки памяти «Найди пару». Есть три уровня сложности. При нажатии на карточку, показывается ее обратная сторона. При нажатии на вторую карточку при совпадении образуется пара, и карточки остаются видны обратной стороной. При несовпадении пара не образуется и обе карточки переворачиваются на лицевую сторону. Необходимо подобрать все пары.

Завершающей активностью является таблица Шульте, направленная на улучшение внимательности и концентрации. Упражнение имеет три режима: классическая таблица с цифрами, буквенная с символами латинского алфавита, а также таблица перемешиваний, где после каждой найденной цифры оставшиеся перемешиваются.

На каждой активности добавлен элемент для возврата к предыдущей активности.

**3.2 База данных**

В ходе анализа потребностей приложения была выбрана облачная база данных Firebase, предлагающая гибкое хранение данных. Рассмотрим разработку базы данных Firebase, ее особенности и преимущества.

Firebase - это облачная платформа, разработанная Google, которая предоставляет различные сервисы для разработки приложений, включая базу данных в реальном времени, которая используется для работы приложения. Firebase позволяет автоматически синхронизировать данные между устройствами и поддерживает множество пользователей одновременно.

Перед тем, как приступить к разработке базы данных, необходимо определить требования приложения и функциональность, которую нужно реализовать. Определим следующие требования:

1. Хранение пользовательских профилей. Необходимо сохранять информацию о каждом пользователе, включая его имя, электронную почту и прогресс тренировок.
2. Хранение наборов коллекций и карточек. Нужно хранить коллекции с карточками, которые пользователи могут создавать и использовать для тренировки.
3. Отслеживание прогресса тренировок. Необходимо записывать результаты тренировок пользователей, чтобы они могли видеть свой прогресс и улучшения.

На основе требований следует приступить к проектированию структуры базы данных. Firebase предоставляет гибкую структуру NoSQL базы данных, основанную на документах, коллекциях и полях. Создадим следующую структуру:

1. Коллекция «Users». Содержит документы, каждый из которых представляет отдельного пользователя и содержит информацию о его профиле.
2. Коллекция «Collections». Содержит документы, каждый из которых представляет отдельный набор коллекций, включая название и поле выбора.
3. Коллекция «Cards». Содержит документы, каждый из которых представляет отдельный набор карточек, включая название и список карточек.
4. Коллекция «Stats». Содержит документы, каждый из которых представляет результаты тренировки пользователя.
5. Коллекция «ShareCollections». Содержит информацию о документах, каждый из которых представляет отдельный набор коллекций от разных пользователей, включая уникальный идентификатор создателя.

Для интеграции Firebase в мобильное приложение, используется Firebase Android SDK для приложения на платформе Android. Необходимо подключить приложение к проекту Firebase, используя уникальный идентификатор проекта и настройки доступа.

Итоговая схема разработанной базы данных представлена на рисунке 3.1.1.

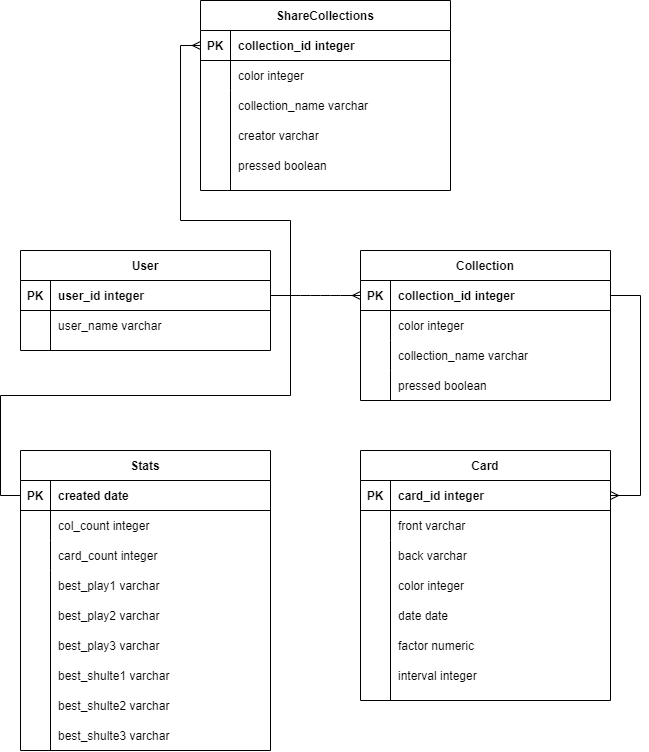


Рисунок 3.1.1 – Схема разработанной базы данных

Правильное проектирование и интеграция базы данных Firebase обеспечивает надежное хранение данных, удобную работу с ними и безопасность пользовательской информации.

**3.3 Реализация приложения**

В ходе разработки приложения был реализован такой функционал, как запоминание информации с помощью карточек и их повторения на основе интервального метода с реализацией алгоритма SuperMemo, интерактив для тренировки памяти в виде игры «Найди пару» и таблицей Шульте.

**3.3.1 Реализация связи с Firebase**

Связь с Firebase реализована в виде отдельного класса FirebaseHelper(). В данном классе определены классы данных UserStat(), Card() и Collection(), являющиеся основной информацией в приложении, определены методы для работы с данными. Реализованные методы представлены в таблице 3.1.

Таблица 3.1 – Методы класса FirebaseHelper

|  |  |
| --- | --- |
| **Название** | **Функция** |
| createCollection() | Создать новую коллекцию |
| getAllCollections() | Получить все существующие коллекции пользователя |
| deleteCollections() | Удалить выбранную коллекцию |
| updateCollection() | Обновить данные выбранной коллекции |
| getCollectionById() | Получить коллекцию с помощью Id |
| getAllCollectionCards() | Получить все карточки в выбранной коллекции |
| getPressedCollectionsIds() | Получить Id отмеченных коллекций |
| addCard() | Добавить новую карточку |
| updateCard() | Обновить данные выбранной карточки |
| deleteCard() | Удалить выбранную карточку |

Продолжение Таблицы 3.1

|  |  |
| --- | --- |
| getAllCards() | Получить все карточки в выбранной коллекции |
| getCardById() | Получить карточку с помощью Id |
| addStat() | Добавить/обновить данные о статистике пользователя |
| getUserStat() | Получить статистику пользователя |
| shareCollection() | Поделиться коллекцией |
| getShareCollections() | Получить коллекции из общего доступа |
| deleteShare() | Удалить коллекцию из общего доступа |
| getShareCollectionsIds | Получить Id коллекций из общего доступа |

Полный код реализации класса FirebaseHelper представлен в приложении А.

**3.3.2 Реализация запоминания информации**

Запоминание информации реализовано с помощью использования карточек, имеющих лицевую и обратную сторону. Методы, отвечающие за действия с карточками и информацией, которую они содержат, находятся в классах EditCardActivity(), RememberCard(). Данные методы представлены в таблицах 3.2, 3.3.

В классе EditCardActivity() представлены методы, которые отвечают за добавление новой карточки или обновление информации уже существующей карточки в соответствующей коллекции.

В классе RememberCard() представлены методы, позволяющие запоминать информацию на карточках с помощью метода интервального повторения, используя алгоритм SuperMemo.

Таблица 3.2– Методы класса EditCardActivity

|  |  |
| --- | --- |
| **Название** | **Функция** |
| addNewCard() | Добавить новую карточку в выбранную коллекцию |
| editCard() | Редактировать уже существующую выбранную карточку из соответствующей коллекции |
| getColorByButton () | Получить цвет для карточки в зависимости от нажатой кнопки цвета |

Таблица 3.3 – Методы класса RememberCard

|  |  |
| --- | --- |
| **Название** | **Функция** |
| theEndToday () | Преобразовать активность в соответствии с окончанием списка карточек |
| updateDate () | Обновить дату следующего повторения карточки |
| calculateInterval () | Рассчитать интервал между датами повторения |
| calculateNextReviewDate () | Рассчитать следующую дату повторения карточки в соответствии с выставленной пользователем оценкой |
| showCardAtIndex () | Отобразить карточку с получаемым индексом |

Полный код представленных классов приведен в приложении А.

**3.3.3 Реализация упражнений для тренировки памяти**

Реализация упражнений для тренировки памяти представлена в виде интерактивной игры под названием «Найди пару» и упражнения «таблица Шульте». Методы, отвечающие за реализацию игры, находятся в классе PlayCardActivity(). В данном классе также находится класс перечисления DifficultyLevel, отвечающий за выбор уровня сложности. Реализованные методы представлены в таблице 3.4.

Таблица 3.4 – Методы класса PlayCardActivity

|  |  |
| --- | --- |
| **Название** | **Функция** |
| onClick () | Проверить образование пары карточек при нажатии |
| FindViews () | Обновить активность в соответствии с выбранным уровнем сложности |
| getCardIndex () | Получить индекс выбранной карточки |
| getCardViewId () | Получить карточку по индексу |
| setGridLayout () | Отобразить карточки в соответствии с выбранным уровнем сложности |
| isCardAlreadyFlipped() | Проверить, перевернута ли карточка |
| flipCard() | Перевернуть карточку лицевой стороной вверх |
| flipCardBack() | Перевернуть карточку обратной стороной вверх |
| checkGameCompletion() | Проверить, закончена ли игра |

Полный код класса PlayCardActivity() приведен в приложении А.

Методы, отвечающие за реализацию упражнения «таблица Шульте» представлены в классе ShulteActivity(). Реализованы три типа таблицы, переключение между ними осуществляется с помощью специальной переменной. Реализованные методы представлены в таблице 3.5.

Таблица 3.5 – Методы класса ShulteActivity

|  |  |
| --- | --- |
| **Название** | **Функция** |
| startGame () | Запустить таймер и начать упражнение |
| shuffleButtons () | Перемешать содержимое внутри GridView |
| showGameFinished () | Показать результат прохождения и остановить таймер |
| resetGame () | Начать упражнение заново |
| onButtonClicked () | Проверить окончание упражнения |

Полный код класса ShulteActivity () приведен в приложении А.

**3.4 Тестирование**

Процесс тестирования любого ресурса включает в себя проверку функциональности, безопасности и производительности с целью выявления ошибок на любом из этапов эксплуатации.

Приложение для запоминания и тренировки памяти было необходимо протестировать по нескольким основным параметрам: соответствие функциональным требованиям и совместимость с разными устройствами.

На рисунке 3.4.1 представлен результат тестирования навигации внутри приложения.

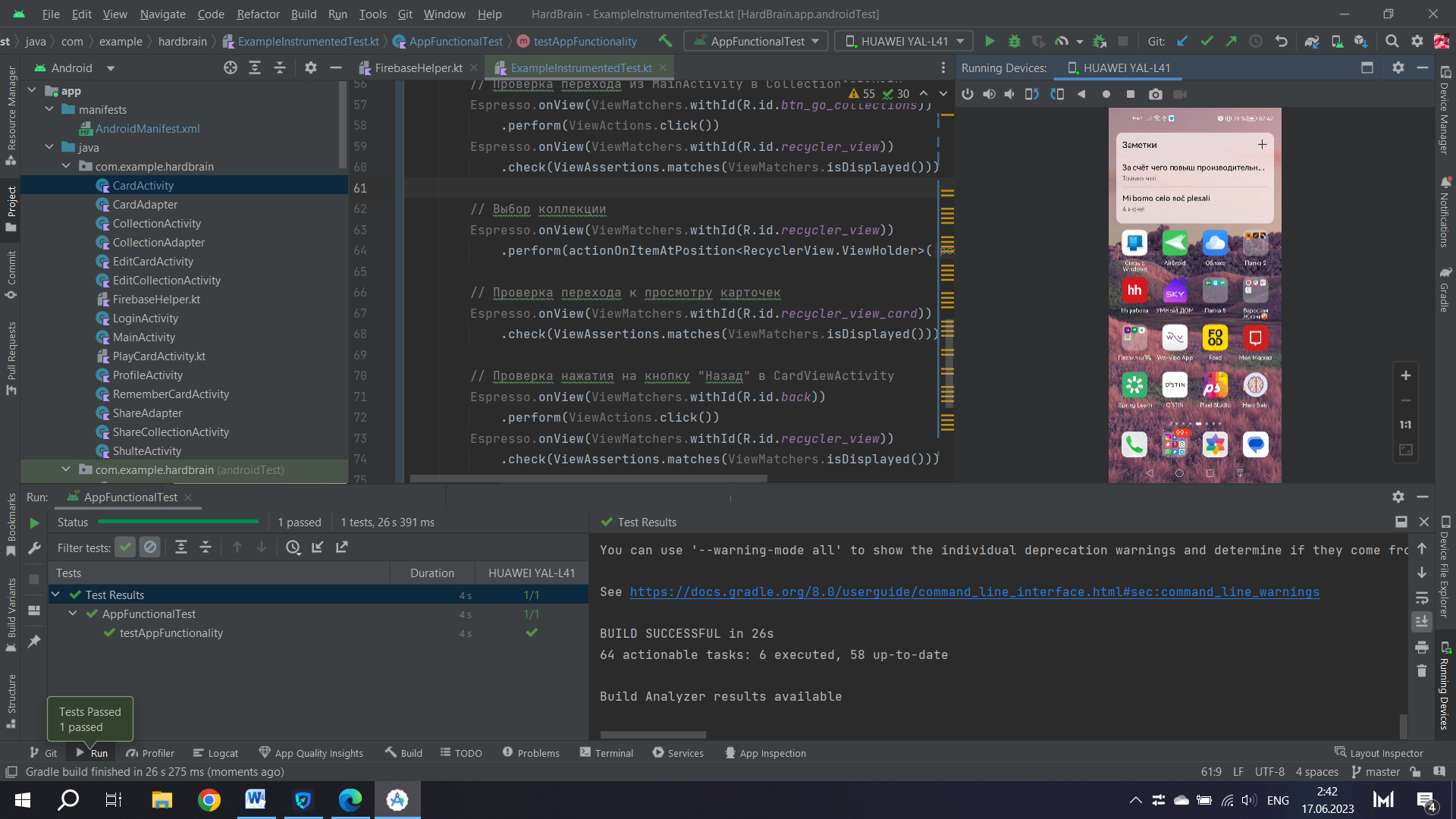


Рисунок 3.4.1 – Результат тестирования

На рисунке 3.4.2 и рисунке 3.4.3 представлена работа приложения на разных устройствах.

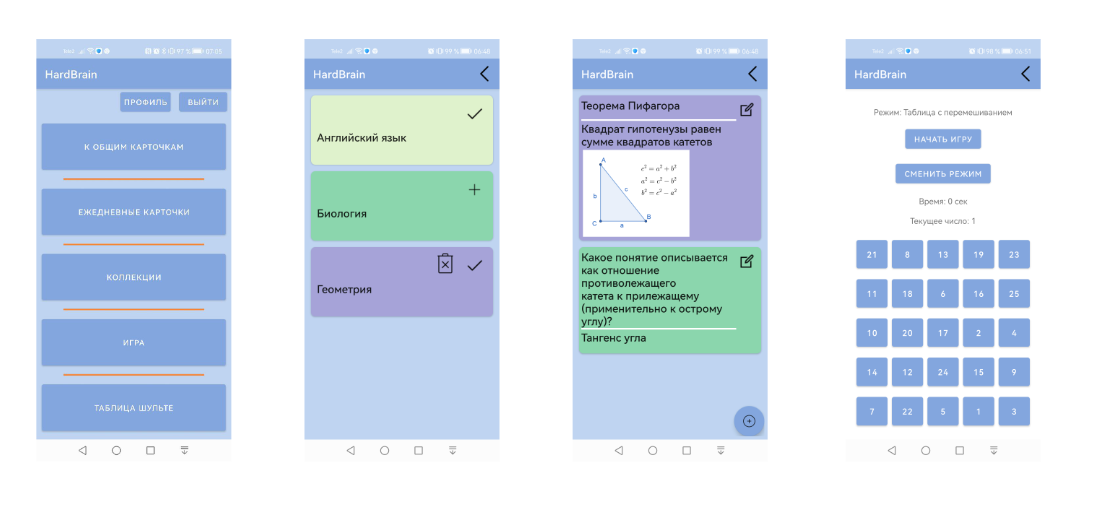


Рисунок 3.4.2 – Работа приложения на Honor 20 Pro

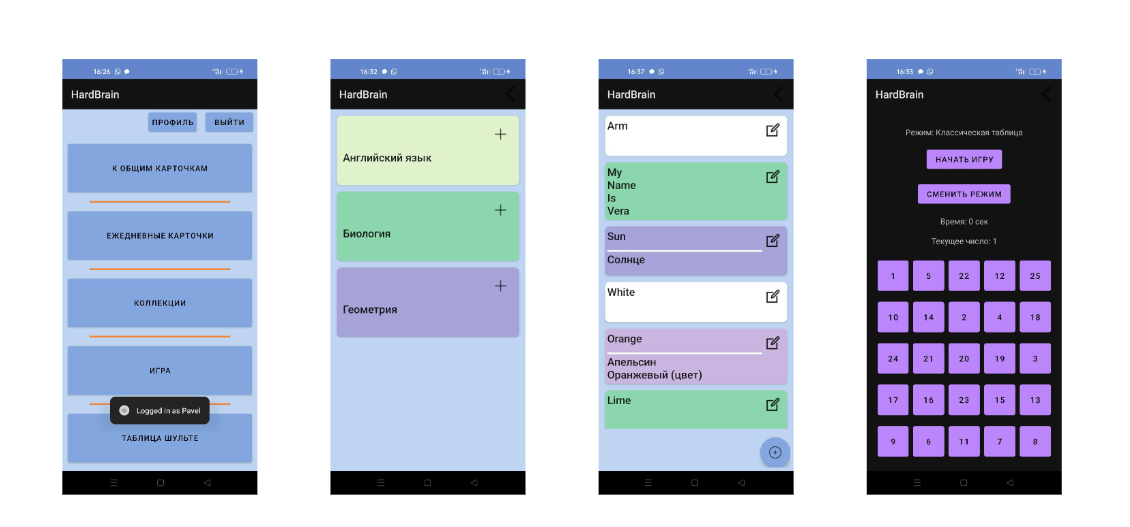


Рисунок 3.4.3 – Работа приложения на Realme 10i

Результаты функционального тестирования представлены в приложении Б.

**Вывод**

В третьем разделе был создан макет и произведена реализация мобильного приложения для запоминания и тренировки памяти, разработана база данных Firebase, а так же произведено тестирование приложения по разным критериям.

**ЗАКЛЮЧЕНИЕ**

В современном мире информация играет значительную роль в жизни человечества. В следствие этого возникает потребность в разработке ресурсов для тренировки памяти и улучшения навыков запоминания информации.

Для достижения поставленной цели в рамках выпускной квалификационной работы были выполнены все поставленные задачи:

1. изучены научные и практические аспекты памяти и ее тренировки;

2. изучен рынок мобильных приложений для тренировки памяти и проанализированы существующие приложения;

3. определены требования к разрабатываемому приложению;

4. разработаны дизайн и интерфейс приложения;

5. разработан функционал приложения;

6. приложение протестировано на разных устройствах;

В ходе анализа предметной области были рассмотрены основные компоненты и понятие памяти, ее виды, классификация и фазы. Были изучены подходы к улучшению памяти и некоторые техники, позволяющие облегчить запоминание информации.

В рамках проектирования приложения были рассмотрены уже существующие аналоги, проанализированы имеющиеся достоинства и недостатки. На основании полученных выводов были определены требования к системе и выбраны технологии разработки и язык программирования.

Результатом работы, проделанной в ходе этапа разработки приложения и связанных с ним компонентов, является готовое мобильное приложение для запоминания и тренировки памяти.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Матвеев, С. Феноменальная память: методы запоминания информации / С. Матвеев. — 2-е изд. — Москва : Альпина Паблишер, 2020. — 160 c. — ISBN 978-5-9614-2274-0. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/93061.html (дата обращения: 12.05.2023).
2. Величковский, Б. Б. Рабочая память человека. Структура и механизмы / Б. Б. Величковский. — Москва : Когито-Центр, 2015. — 247 c. — ISBN 978-5-89353-467-2. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/51950.html (дата обращения: 10.05.2023).
3. CogniFit research [Электронный ресурс] – URL: <https://www.cognifit.com/ru/memory>. – Дата обращения: 15.05.2021
4. Developers [Электронный ресурс] – URL: https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies.html. – Дата обращения: 16.05.2021
5. Контур | Журнал [Электронный ресурс] – URL: https://kontur.ru/articles/5140. – Дата обращения: 16.05.2021
6. Peaasi.ee [Электронный ресурс] – URL: https://peaasi.ee/ru/%D0%BF%D1%80%D0%B8%D0%B5%D0%BC%D1%8B-%D0%B7%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D0%BD%D0%B8%D1%8F/#1589190774733-3ff503c8-acd7. – Дата обращения: 18.05.2021
7. Kotlin Documentation [Электронный ресурс] – URL: <https://kotlinlang.org/docs/home.html> - Дата обращения: 18.05.2021
8. Сомон., П. -И. Волшебство Kotlin / П. -И. Сомон. ; перевод А. Н. Киселев. — Москва : ДМК Пресс, 2020. — 536 c. — ISBN 978-5-97060-801-2.
9. Kotlin. Программирование для профессионалов. — СПб.: Питер, 2020. — 464 с.: ил. — (Серия «Для профессионалов»). ISBN 978-5-4461- 1243-2
10. Жемеров, Д. Kotlin в действии / Д. Жемеров, С. Исакова ; перевод А. Н. Киселев. — Москва : ДМК Пресс, 2018. — 402 c. — ISBN 978-5-97060-497-7
11. Аделекан И. Kotlin: программирование на примерах: Пер. с англ. – СПб.: БХВ-Петербург, 2020. – 432 с.: ил. – ISBN 978-5-9775-6673-5

**ПРИЛОЖЕНИЕ А**

**FirebaseHelper.kt**

package com.example.hardbrain  
import …  
data class Collection(  
 var id: String? = null,  
 var name: String = "",  
 var pressed: Boolean = false,  
 var color: Int = -1, // цвет по умолчанию white  
 var cards: HashMap<String, Card> = hashMapOf(),  
 var creator: String? = null  
) : Serializable { constructor() : this(null, "",false, -1, hashMapOf(), null) }  
data class UserStat(  
 var created: String = "",  
 var col\_count: String? = null,  
 var card\_count: String? = null,  
 var best\_play1: String? = null,  
 var best\_play2: String? = null,  
 var best\_play3: String? = null,  
 var best\_shulte1: String? = null,  
 var best\_shulte2: String? = null,  
 var best\_shulte3: String? = null  
): Serializable  
data class Card(  
 var id: String? = null,  
 var front: String = "",  
 var back: String = "",  
 var date: String? = "",  
 var collectionId: String? = "",  
 var interval: Int = 1, //по умолчанию 0 дней  
 var factor: Double = 1.0, //изначально 1  
 var color: Int = -1, // цвет по умолчанию white  
 var imageUrl\_f: String? = null,  
 var imageUrl\_b: String? = null  
): Serializable  
class FirebaseHelper {  
 private val database: FirebaseDatabase = FirebaseDatabase.getInstance()  
 private val auth: FirebaseAuth = FirebaseAuth.getInstance()  
 private val userId: String? = auth.currentUser?.uid  
 private val collectionsRef: DatabaseReference =  
 database.getReference("users/$userId/collections")  
 private val userStatsRef: DatabaseReference = database.getReference("users/$userId/stats")  
 private val shareRef: DatabaseReference = database.getReference("shareCollections")  
 val userStatRef = userStatsRef.child(userId.toString())  
 private val storage: FirebaseStorage = FirebaseStorage.getInstance()  
 fun uploadImageToFirestore(imageUri: Uri, collectionId: String, cardId: String, callback: (imageUrl: String?) -> Unit) {  
 val storageRef = storage.reference  
 val imageRef = storageRef.child("images/$collectionId/$cardId.jpg")  
 val uploadTask = imageRef.putFile(imageUri)  
 uploadTask.continueWithTask { task ->  
 if (!task.isSuccessful) {  
 task.exception?.let {  
 throw it } }  
 imageRef.downloadUrl  
 }.addOnCompleteListener { task ->  
 if (task.isSuccessful) {  
 val imageUrl = task.result.toString()  
 callback(imageUrl)  
 } else {  
 callback(null) } } }  
 fun getCollectionsCount(callback: (Long) -> Unit) {  
 collectionsRef.addListenerForSingleValueEvent(object : ValueEventListener { override fun onDataChange(snapshot: DataSnapshot) {  
 val collectionsCount = snapshot.childrenCount  
 callback(collectionsCount) }  
 override fun onCancelled(error: DatabaseError) {  
 Log.e("Firebase", "Error fetching user stat", error.toException())  
 callback(0) }

})

}  
 fun getCardsCount(callback: (Long) -> Unit){  
 collectionsRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 var cardsCount = 0L  
 for (collectionSnapshot in snapshot.children) {  
 val cardsSnapshot = collectionSnapshot.child("cards")  
 cardsCount += cardsSnapshot.childrenCount  
 }  
 callback(cardsCount)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.e("Firebase", "Error fetching user stat", error.toException())  
 callback(0) }  
 })  
 }  
 fun addStat(userStat: UserStat, callback: (Boolean) -> Unit){  
 userStatsRef.setValue(userStat)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { exception ->  
 Log.e("Firebase", "Error adding collection", exception)  
 callback(false) }  
 }  
 fun getUserStat(callback: (UserStat?) -> Unit) {  
 userStatsRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val userStat = snapshot.getValue(UserStat::class.java)  
 callback(userStat)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.e("Firebase", "Error fetching user stat", error.toException())  
 callback(null) }  
 })  
 }  
 fun shareCollection(collection: Collection, callback: (Boolean) -> Unit) { shareRef.child(collection.id.toString()).setValue(collection)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { exception ->  
 Log.e("Firebase", "Error adding collection", exception)  
 callback(false) }  
 }  
 fun getShareCollections(callback: (MutableList<Collection>) -> Unit) {  
 shareRef.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val collections = mutableListOf<Collection>()  
 snapshot.children.forEach { child ->  
 val collection = child.getValue(Collection::class.java)  
 collection?.id = child.key  
 collection?.let { collections.add(it) }  
 }  
 callback(collections)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 // Обработка ошибки }  
 })  
 }  
 fun deleteShare(collectionId: String, callback: (Boolean) -> Unit) {  
 shareRef.child(collectionId).removeValue()  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { callback(false) }  
 }  
 fun getShareCollectionsIds(callback: (List<String>) -> Unit) {  
 shareRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 val pressedCollectionsIds = mutableListOf<String>()  
 for (collectionSnapshot in dataSnapshot.children) {  
 val collectionId = collectionSnapshot.key  
 collectionId?.let { pressedCollectionsIds.add(it) }  
 }  
 callback(pressedCollectionsIds)  
 }  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Обработка ошибки получения данных  
 } }) }  
 fun getShareCards(collectionId: String, callback: (MutableList<Card>) -> Unit) { getCardsRefBySharedCollectionId(collectionId).addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val cards = mutableListOf<Card>()  
 snapshot.children.forEach { child ->  
 val card = child.getValue(Card::class.java)  
 card?.id = child.key  
 card?.let { cards.add(it) }  
 }  
 callback(cards)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.w(TAG, "loadCards:onCancelled", error.toException())  
 Log.d("FirebaseHelper", "cards: users/$userId/cards")  
 }  
 })  
 }  
 fun createCollection(collection: Collection, callback: (Boolean) -> Unit) {  
 val newCollectionRef = collectionsRef.push() // Создаем ссылку на новый узел сгенерированным методом push()  
 val collectionId = newCollectionRef.key // Получаем сгенерированный идентификатор из ссылки  
 collection.id = collectionId // Устанавливаем идентификатор в поле id объекта collection  
 newCollectionRef.setValue(collection)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { exception ->  
 Log.e("Firebase", "Error adding collection", exception)  
 callback(false) }  
 }  
 fun addCollection(collection: Collection, callback: (Boolean) -> Unit) { collectionsRef.child(collection.id.toString()).setValue(collection)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { exception ->  
 Log.e("Firebase", "Error adding collection", exception)  
 callback(false) }  
 }  
 fun getAllCollections(callback: (MutableList<Collection>) -> Unit) {  
 collectionsRef.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val collections = mutableListOf<Collection>()  
 snapshot.children.forEach { child ->  
 val collection = child.getValue(Collection::class.java)  
 collection?.id = child.key  
 collection?.let { collections.add(it) }  
 }  
 callback(collections)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 // Обработка ошибки  
 }  
 })  
 }  
 fun deleteCollections(collectionId: String, callback: (Boolean) -> Unit) {  
 collectionsRef.child(collectionId).removeValue()  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { callback(false) }  
 }  
 fun updateCollection(collection: Collection, callback: (Boolean) -> Unit) {  
 collection.id?.let {  
 collectionsRef.child(it).setValue(collection)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { callback(false) }  
 }  
 }  
 fun getCollectionById(collectionId: String, callback: (Collection?) -> Unit) { collectionsRef.child(collectionId).addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val collection = snapshot.getValue(Collection::class.java)  
 callback(collection)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.w(TAG, "loadCard:onCancelled", error.toException())  
 Log.d("FirebaseHelper", "cards: users/$userId/cards")  
 }  
 })  
 }  
 fun getCollectionsIds(callback: (List<String>) -> Unit) {  
 collectionsRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 val pressedCollectionsIds = mutableListOf<String>()  
 for (collectionSnapshot in dataSnapshot.children) {  
 val collectionId = collectionSnapshot.key  
 collectionId?.let { pressedCollectionsIds.add(it) }  
 }  
 callback(pressedCollectionsIds)  
 }  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Обработка ошибки получения данных  
 }  
 })  
 }  
 fun getAllCollectionCards(callback: (List<Card>) -> Unit) {  
 collectionsRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 val allCards = mutableListOf<Card>()  
 for (collectionSnapshot in dataSnapshot.children) {  
 val cardsSnapshot = collectionSnapshot.child("cards")  
 for (cardSnapshot in cardsSnapshot.children){  
 val card = cardSnapshot.getValue(Card::class.java)  
 card?.let { allCards.add(it) }  
 }  
 }  
 callback(allCards)  
 }  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Обработка ошибки   
 }  
 })  
 }  
 fun getPressedCollectionsIds(callback: (List<String>) -> Unit) {  
 collectionsRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 val pressedCollectionsIds = mutableListOf<String>()  
 for (collectionSnapshot in dataSnapshot.children) {  
 val isPressed = collectionSnapshot.child("pressed").getValue(Boolean::class.java)  
 if (isPressed == true) {  
 val collectionId = collectionSnapshot.key  
 collectionId?.let { pressedCollectionsIds.add(it) }  
 }  
 }  
 callback(pressedCollectionsIds)  
 }  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Обработка ошибки получения данных  
 }  
 })  
 }  
 fun addCard(card: Card, collectionId: String, callback: (Boolean) -> Unit) {  
 val newCardRef = getCardsRefByCollectionId(collectionId).push()   
 val cardId = newCardRef.key   
 card.id = cardId   
 newCardRef.setValue(card)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { exception ->  
 Log.e("Firebase", "Error adding card", exception)  
 callback(false) }  
 }  
 fun updateCard(card: Card, collectionId: String, callback: (Boolean) -> Unit) {  
 card.id?.let { getCardsRefByCollectionId(collectionId).child(it).setValue(card)  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { callback(false) }  
 }  
 }  
 fun deleteCard(cardId: String, collectionId: String, callback: (Boolean) -> Unit) { getCardsRefByCollectionId(collectionId).child(cardId).removeValue()  
 .addOnSuccessListener { callback(true) }  
 .addOnFailureListener { callback(false) }  
 }  
 fun getAllCards(collectionId: String, callback: (MutableList<Card>) -> Unit) { getCardsRefByCollectionId(collectionId).addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val cards = mutableListOf<Card>()  
 snapshot.children.forEach { child ->  
 val card = child.getValue(Card::class.java)  
 card?.id = child.key  
 card?.let { cards.add(it) }  
 }  
 callback(cards)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.w(TAG, "loadCards:onCancelled", error.toException())  
 Log.d("FirebaseHelper", "cards: users/$userId/cards")  
 }  
 })  
 }  
 fun getCardById(cardId: String, collectionId: String, callback: (Card?) -> Unit) { getCardsRefByCollectionId(collectionId).child(cardId).addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val card = snapshot.getValue(Card::class.java)  
 callback(card)  
 }  
 override fun onCancelled(error: DatabaseError) {  
 Log.w(TAG, "loadCard:onCancelled", error.toException())  
 Log.d("FirebaseHelper", "cards: users/$userId/cards")  
 } }) }  
 fun getCardsRefByCollectionId(collectionId: String): DatabaseReference {  
 return database.getReference("users/$userId/collections/$collectionId/cards") }  
 fun getCardsRefBySharedCollectionId(collectionId: String): DatabaseReference {  
 return database.getReference("shareCollections/$collectionId/cards")  
 }  
}

**RememberCardActivity.kt**

package com.example.hardbrain  
import …  
class RememberCardActivity: AppCompatActivity() {  
 private lateinit var cardContainer: LinearLayout  
 private lateinit var frontCard: FrameLayout  
 private lateinit var backCard: FrameLayout  
 private lateinit var frontText: TextView  
 private lateinit var backText: TextView  
 private lateinit var frontImage: ImageView  
 private lateinit var backImage: ImageView  
 private lateinit var btnOne: Button  
 private lateinit var btnTwo: Button  
 private lateinit var btnThree: Button  
 private lateinit var btnFour: Button  
 private lateinit var btnFive: Button  
 private lateinit var firebaseHelper: FirebaseHelper  
 private var isShowingFrontCard = true  
 private var urlF: String? = null  
 private var urlB: String? = null  
 @RequiresApi(Build.VERSION\_CODES.O)  
 @SuppressLint("MissingInflatedId")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.remember\_card)  
 firebaseHelper = FirebaseHelper()  
 val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")  
 // Переменная для хранения текущего индекса карточки  
 var currentIndex = 0  
 var rating = 0  
 val currentDate = LocalDate.now() // Получаем текущую дату  
 cardContainer = findViewById(R.id.card\_container)  
 frontCard = findViewById(R.id.front\_card)  
 backCard = findViewById(R.id.back\_card)  
 btnOne = findViewById(R.id.btnOne)  
 btnTwo = findViewById(R.id.btnTwo)  
 btnThree = findViewById(R.id.btnThree)  
 btnFour = findViewById(R.id.btnFour)  
 btnFive = findViewById(R.id.btnFive)  
 frontText = findViewById(R.id.txtFrontSide)  
 backText = findViewById(R.id.txtBackSide)  
 frontImage = findViewById(R.id.image\_view\_f)  
 backImage = findViewById(R.id.image\_view\_b)  
  
 val flipToFrontAnimation = AnimationUtils.loadAnimation(this, R.anim.flip\_to\_front)  
 val flipToBackAnimation = AnimationUtils.loadAnimation(this, R.anim.flip\_to\_back)  
 firebaseHelper.getAllCollectionCards { allCards ->  
 Log.d("cards", allCards.toString())  
 val cardsDate = allCards.filter { card ->  
 val dateString = card.date  
 val date = LocalDate.parse(dateString, formatter)  
 date <= currentDate  
 }  
 firebaseHelper.getPressedCollectionsIds { pressedCollectionsIds ->  
 Log.d("pressedCollectionsIds", pressedCollectionsIds.toString())  
 val cards = cardsDate.filter { card -> pressedCollectionsIds.contains(card.collectionId)  
 }  
 Log.d("cardsPressed", cards.toString())  
 if (cards.isNotEmpty()) {  
 // Показать первую карточку при запуске активности  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 // Обработчик нажатия кнопки "0"  
 btnOne.setOnClickListener {  
 rating = 1

updateDate(cards[currentIndex], formatter, rating)  
 currentIndex++  
 if (currentIndex < cards.size) {  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 }  
 // Обработчик нажатия кнопки "2"  
 btnTwo.setOnClickListener {  
 rating = 2  
 updateDate(cards[currentIndex], formatter, rating)  
 currentIndex++  
 if (currentIndex < cards.size) {  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 }  
 // Обработчик нажатия кнопки "3"  
 btnThree.setOnClickListener {  
 rating = 3  
 updateDate(cards[currentIndex], formatter, rating)  
 currentIndex++  
 if (currentIndex < cards.size) {  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 }  
 // Обработчик нажатия кнопки "4"  
 btnFour.setOnClickListener {  
 rating = 4  
 updateDate(cards[currentIndex], formatter, rating)  
 currentIndex++  
 if (currentIndex < cards.size) {  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 }  
 // Обработчик нажатия кнопки "5"  
 btnFive.setOnClickListener {  
 rating = 5  
 updateDate(cards[currentIndex], formatter, rating)  
 currentIndex++  
 if (currentIndex < cards.size) {  
 showCardAtIndex(currentIndex, cards)  
 } else {  
 theEndToday()  
 }  
 }  
 }  
 }  
 cardContainer.setOnClickListener {  
 if (isShowingFrontCard) {  
 frontCard.startAnimation(flipToBackAnimation)  
 frontCard.visibility = View.GONE  
 backCard.visibility = View.VISIBLE  
 if (urlB != null) {  
 Log.d("urlB", urlB.toString())  
 Picasso.get().load(urlB).into(backImage)  
 backImage.visibility = View.VISIBLE  
 } else {  
 backImage.visibility = View.GONE  
 }  
 } else {  
 backCard.startAnimation(flipToFrontAnimation)  
 backCard.visibility = View.GONE  
 frontCard.visibility = View.VISIBLE  
 if (urlF != null) {  
 Log.d("urlF", urlF.toString())  
 Picasso.get().load(urlF).into(frontImage)  
 frontImage.visibility = View.VISIBLE  
 } else {  
 frontImage.visibility = View.GONE  
 }  
 }  
 isShowingFrontCard = !isShowingFrontCard  
 }  
 }  
  
 // Функция для отображения карточки по указанному индексу  
 fun showCardAtIndex(index: Int, cards: List<Card>) {  
 val currentCard = cards[index]  
 // Отобразить карточку на экране  
 frontText.text = currentCard.front  
 backText.text = currentCard.back  
 urlF = currentCard.imageUrl\_f  
 urlB = currentCard.imageUrl\_b  
 }  
 @RequiresApi(Build.VERSION\_CODES.O)  
 fun calculateNextReviewDate(previousDate: LocalDate, interval: Long): LocalDate {  
 return previousDate.plusDays(interval)  
 }  
 fun calculateInterval(previousInterval: Long, factor: Double, rating: Int): Long {  
 val exponent = rating - 1  
 val newInterval = previousInterval \* factor.pow(exponent)  
 return newInterval.toLong()  
 }  
 @RequiresApi(Build.VERSION\_CODES.O)  
 fun updateDate(card: Card, formatter: DateTimeFormatter, rating: Int){  
 val previousInterval = card.interval.toLong()  
 var factor = card.factor //1.0  
 Log.d("factor", factor.toString())  
 Log.d("id", card.id.toString())  
 Log.d("collectionId", card.collectionId.toString())  
 if (rating == 1 || rating == 2){  
 factor = 1.0  
 }  
 else if (rating == 4){  
 factor += 0.15  
 }  
 else if (rating == 5){  
 factor += 0.3  
 }  
 val previousDateString = card.date  
 Log.d("previousDateString", previousDateString.toString())  
 val previousDate = LocalDate.parse(previousDateString, formatter)  
 Log.d("previousDate", previousDate.toString())  
 val newInterval = calculateInterval(previousInterval, factor, rating)  
 val nextReviewDate = calculateNextReviewDate(previousDate, newInterval)  
 val nextReviewDateString = nextReviewDate.format(formatter)  
 val editedCard = card  
 editedCard.date = nextReviewDateString  
 editedCard.interval = newInterval.toInt()  
 editedCard.factor = 1.0  
 card.collectionId?.let {  
 firebaseHelper.updateCard(editedCard, it) { success ->  
 if (success) {  
 Log.d("updateDateCard", nextReviewDateString)  
 } else {  
 Toast.makeText(this,"Ошибка при редактировании карточки", Toast.LENGTH\_SHORT).show()  
 }  
 }  
 }  
 }  
 fun theEndToday(){  
 frontText.text = "Сегодня Вы повторили все карточки! Приходите завтра:)"  
 backText.text = "Сегодня Вы повторили все карточки! Приходите завтра:)"  
 btnOne.visibility = View.GONE  
 btnTwo.visibility = View.GONE  
 btnThree.visibility = View.GONE  
 btnFour.visibility = View.GONE  
 btnFive.visibility = View.GONE  
 }  
 override fun onCreateOptionsMenu(menu: Menu): Boolean {  
 menuInflater.inflate(R.menu.my\_menu, menu)  
 return true  
 }  
 override fun onPrepareOptionsMenu(menu: Menu): Boolean {  
 val deleteItem = menu.findItem(R.id.delete)  
 deleteItem.isVisible = false  
 return super.onPrepareOptionsMenu(menu)  
 }  
 fun updateOptionsMenu() {  
 invalidateOptionsMenu()  
 }  
 public fun onBackClick(item: MenuItem){  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 updateOptionsMenu()  
 }  
 public fun onDeleteSelectedCardsButtonClick(item: MenuItem){}  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 return when (item.itemId) {  
 R.id.back -> {  
 // Обработка нажатия кнопки "Назад"  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 true  
 }  
 else -> super.onOptionsItemSelected(item)  
 }  
 }  
}

**EditCardActivity.kt**

package com.example.hardbrain  
import …  
class EditCardActivity : AppCompatActivity() {  
 private lateinit var adapter: CardAdapter  
 private lateinit var firebaseHelper: FirebaseHelper  
 private lateinit var mContext: Context  
 private lateinit var btnColorWhite: Button  
 private lateinit var btnColorBlue: Button  
 private lateinit var btnColorRed: Button  
 private lateinit var btnColorGreen: Button  
 private lateinit var btnColorYellow: Button  
 private lateinit var collectionId: String  
 private lateinit var cardId: String  
 private var imageUri: Uri? = null  
 private var imageUrl: String? = null  
 private var isImageF = true  
 companion object {  
 private const val REQUEST\_IMAGE\_PICK = 4  
 }  
 @RequiresApi(Build.VERSION\_CODES.O)  
 @SuppressLint("SuspiciousIndentation")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_edit\_card)  
 val isNewCard = intent.getBooleanExtra("isNewCard", true)  
 cardId = intent.getStringExtra("card\_id").toString()  
 val position = intent.getIntExtra("position", -1)  
 collectionId = intent.getStringExtra("collectionId")!!  
 mContext = this // инициализация контекста  
 adapter = CardAdapter(mutableListOf(), collectionId)  
 firebaseHelper = FirebaseHelper()  
 var color: Int = ContextCompat.getColor(this, R.color.white) // цвет по умолчанию  
 val NowDate = LocalDate.now()  
 val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")  
 val dateString = NowDate.format(formatter)  
 btnColorWhite = findViewById(R.id.btn\_color\_white)  
 btnColorBlue = findViewById(R.id.btn\_color\_blue)  
 btnColorRed = findViewById(R.id.btn\_color\_red)  
 btnColorGreen = findViewById(R.id.btn\_color\_green)  
 btnColorYellow = findViewById(R.id.btn\_color\_yellow)  
 val etFront = findViewById<TextView>(R.id.edit\_text\_front)  
 val etBack = findViewById<TextView>(R.id.edit\_text\_back)  
 val btnSave = findViewById<TextView>(R.id.button\_save\_card)  
 val btnAddImage1 = findViewById<ImageButton>(R.id.add\_image\_1)  
 val btnAddImage2 = findViewById<ImageButton>(R.id.add\_image\_2)  
 btnColorWhite.setOnClickListener { color = getColorByButton(it, this) }  
 btnColorBlue.setOnClickListener { color = getColorByButton(it, this) }  
 btnColorRed.setOnClickListener { color = getColorByButton(it, this) }  
 btnColorGreen.setOnClickListener { color = getColorByButton(it, this) }  
 btnColorYellow.setOnClickListener { color = getColorByButton(it, this) }  
 btnAddImage1.setOnClickListener {  
 isImageF = true  
 openGallery()  
 }  
 btnAddImage2.setOnClickListener {  
 isImageF = false  
 openGallery()  
 }  
 if (isNewCard) {  
 // Обработчик нажатия на кнопку сохранения карточки  
 btnSave.setOnClickListener {  
 // Считываем данные из полей формы  
 val front = etFront.text.toString().trim()  
 val back = etBack.text.toString().trim()  
 // Проверяем, что оба поля заполнены  
 if (front.isEmpty() || back.isEmpty()) {  
 Toast.makeText(this, "Заполните все поля", Toast.LENGTH\_SHORT).show()  
 return@setOnClickListener  
 }  
 // Создаем новую карточку  
 val newCard = Card(UUID.randomUUID().toString(), front, back, dateString, collectionId, 1, 1.0, color)  
 if (isImageF){  
 if (imageUrl != null){  
 newCard.imageUrl\_f = imageUrl  
 }  
 }  
 else{  
 if (imageUrl != null){  
 newCard.imageUrl\_b = imageUrl  
 }  
 }  
 firebaseHelper.addCard(newCard, collectionId) { success ->  
 if (success) {  
 adapter.cards.add(newCard) adapter.notifyItemInserted(adapter.cards.size - 1)  
 val intent = Intent(this, CardActivity::class.java)  
 intent.putExtra("collectionId", collectionId)  
 startActivity(intent)  
 finish()  
 } else {  
 Toast.makeText(this, "Ошибка при добавлении карточки", Toast.LENGTH\_SHORT).show()  
 }  
 }  
 }  
 }  
 else{  
 cardId?.let {  
 firebaseHelper.getCardById(it, collectionId) { card ->  
 if (card != null) {  
 // Редактируем существующую карточку  
 etFront.text = card.front  
 etBack.text = card.back  
 color = card.color  
 Log.d("Id", cardId)  
 Log.d("Position", position.toString())  
 btnSave.setOnClickListener {  
 val front = etFront.text.toString().trim()  
 val back = etBack.text.toString().trim()  
 // Проверяем, что оба поля заполнены  
 if (front.isEmpty() || back.isEmpty()) {  
 Toast.makeText(this, "Заполните все поля", Toast.LENGTH\_SHORT).show()  
 return@setOnClickListener  
 }  
 val editedCard = Card(cardId, front, back, card.date, card.collectionId, card.interval, card.factor, color, card.imageUrl\_f, card.imageUrl\_b)  
 if (isImageF){  
 if (imageUrl != null){  
 editedCard.imageUrl\_f = imageUrl  
 }  
 }  
 else{  
 if (imageUrl != null){  
 editedCard.imageUrl\_b = imageUrl  
 }  
 }  
 Log.d("Id editedCard", editedCard.id.toString())  
 Log.d("collectionId", collectionId) firebaseHelper.updateCard(editedCard, collectionId) { success ->  
 if (success) { adapter.notifyItemChanged(position)  
 val intent = Intent(this,CardActivity::class.java)

intent.putExtra("collectionId", collectionId)  
 startActivity(intent)  
 finish()  
 } else {  
 Toast.makeText(this,"Ошибка при редактировании карточки",Toast.LENGTH\_SHORT).show()  
 }  
 }  
 }  
 }  
 else {  
 Toast.makeText(this,"Карточка с таким идентификатором не найдена",Toast.LENGTH\_SHORT).show()  
 }  
 }  
 }  
 }  
 }  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if (requestCode == REQUEST\_IMAGE\_PICK && resultCode == Activity.RESULT\_OK && data != null) {  
 imageUri = data.data  
 imageUri?.let {  
 firebaseHelper.uploadImageToFirestore(it, collectionId, cardId) { imageUrl ->  
 if (imageUrl != null) {  
 this.imageUrl = imageUrl  
 } else { }  
 }  
 }  
 }  
 }  
 private fun openGallery() {  
 val intent = Intent(Intent.ACTION\_PICK, MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI)  
 startActivityForResult(intent, REQUEST\_IMAGE\_PICK)  
 }  
 fun getColorByButton(view: View, context: Context): Int {  
 return when (view.id) {  
 R.id.btn\_color\_blue -> ContextCompat.getColor(context, R.color.blue)  
 R.id.btn\_color\_red -> ContextCompat.getColor(context, R.color.red)  
 R.id.btn\_color\_green -> ContextCompat.getColor(context, R.color.green)  
 R.id.btn\_color\_yellow -> ContextCompat.getColor(context, R.color.yellow)  
 else -> ContextCompat.getColor(context, R.color.white) // цвет по умолчанию  
 }  
 }  
 override fun onCreateOptionsMenu(menu: Menu): Boolean {  
 menuInflater.inflate(R.menu.my\_menu, menu)  
 return true  
 }  
 override fun onPrepareOptionsMenu(menu: Menu): Boolean {  
 val deleteItem = menu.findItem(R.id.delete)  
 deleteItem.isVisible = false  
 return super.onPrepareOptionsMenu(menu)  
 }  
 fun updateOptionsMenu() {  
 invalidateOptionsMenu()  
 }  
 public fun onBackClick(item: MenuItem){  
 super.onBackPressed()  
 updateOptionsMenu()  
 }  
 public fun onDeleteSelectedCardsButtonClick(item: MenuItem){}  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 return when (item.itemId) {  
 R.id.back -> {  
 super.onBackPressed() // Обработка нажатия кнопки "Назад"  
 true  
 }  
 else -> super.onOptionsItemSelected(item)  
 }  
 }  
}

**PlayCardActivity.kt**

package com.example.hardbrain  
import …  
enum class DifficultyLevel {  
 EASY,  
 MEDIUM,  
 HARD  
}  
class PlayCardActivity: AppCompatActivity(), View.OnClickListener {  
 private val cardImagesEasy = listOf(  
 R.drawable.card\_image\_1,  
 R.drawable.card\_image\_2,  
 R.drawable.card\_image\_3,  
 R.drawable.card\_image\_4,  
 R.drawable.card\_image\_5,  
 R.drawable.card\_image\_6  
 )  
 private val cardImagesMedium = listOf(  
 R.drawable.card\_image\_1,  
 R.drawable.card\_image\_2,  
 R.drawable.card\_image\_3,  
 R.drawable.card\_image\_4,  
 R.drawable.card\_image\_5,  
 R.drawable.card\_image\_6,  
 R.drawable.card\_image\_7,  
 R.drawable.card\_image\_8,  
 R.drawable.card\_image\_9  
 )  
 private val cardImagesHard = listOf(  
 R.drawable.card\_image\_1,  
 R.drawable.card\_image\_2,  
 R.drawable.card\_image\_3,  
 R.drawable.card\_image\_4,  
 R.drawable.card\_image\_5,  
 R.drawable.card\_image\_6,  
 R.drawable.card\_image\_7,  
 R.drawable.card\_image\_8,  
 R.drawable.card\_image\_9,  
 R.drawable.card\_image\_10,  
 R.drawable.card\_image\_11,  
 R.drawable.card\_image\_12  
 )  
 private var cardPairs = cardImagesEasy.size  
 private val cards = mutableListOf<Int>()  
 private var flippedCardsCount = 0  
 private var firstCardIndex = -1  
 private var secondCardIndex = -1  
 private var cardCount = 0  
 private lateinit var gridLayoutEasy: GridLayout  
 private lateinit var gridLayoutMedium: GridLayout  
 private lateinit var gridLayoutHard: GridLayout  
 private lateinit var difficultyLevel: DifficultyLevel  
 private lateinit var imageViews: List<ImageView>  
 private lateinit var timerTextView: TextView  
 private lateinit var firebaseHelper: FirebaseHelper  
 private val timerHandler = Handler()  
 private var elapsedTime: Long = 0  
 private val timerRunnable = object : Runnable {  
 override fun run() {  
 elapsedTime += 1000  
 val seconds = elapsedTime / 1000  
 timerTextView.text = "Время: $seconds сек"  
 timerHandler.postDelayed(this, 1000)  
 }  
 }  
 @SuppressLint("MissingInflatedId")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.play\_card)  
 gridLayoutEasy = findViewById(R.id.gridLayoutEasy)  
 gridLayoutMedium = findViewById(R.id.gridLayoutMedium)  
 gridLayoutHard = findViewById(R.id.gridLayoutHard)  
 timerTextView = findViewById(R.id.timer)  
 firebaseHelper = FirebaseHelper()  
 difficultyLevel = DifficultyLevel.EASY  
 setGridLayout(difficultyLevel)  
 cards.addAll(cardImagesEasy)  
 cards.addAll(cardImagesEasy)  
 cards.shuffle()  
 val btEasy = findViewById<Button>(R.id.bt\_easy)  
 val btMedium = findViewById<Button>(R.id.bt\_medium)  
 val btHard = findViewById<Button>(R.id.bt\_hard)  
 val btTimer = findViewById<Button>(R.id.bt\_timer)  
 imageViews = FindViews(this, difficultyLevel)  
 imageViews.forEach{element ->  
 element.setOnClickListener(this)  
 // Изображения для каждой карточки  
 val index = getCardIndex(element, difficultyLevel)  
 element.setImageResource(cards[index])  
 flipCardBack(element, getCardIndex(element, difficultyLevel))  
 }  
 // Установка уровня сложности  
 btEasy.setOnClickListener{  
 restartTimer()  
 timerHandler.removeCallbacks(timerRunnable)  
 difficultyLevel = DifficultyLevel.EASY  
 setGridLayout(difficultyLevel)  
 cardPairs = cardImagesEasy.size  
 cards.clear()  
 // Перемешать картинки  
 cards.addAll(cardImagesEasy)  
 cards.addAll(cardImagesEasy)  
 cards.shuffle()  
 imageViews = FindViews(this, difficultyLevel)  
 imageViews.forEach{element ->  
 element.setOnClickListener(this)  
 val index = getCardIndex(element, difficultyLevel)  
 element.setImageResource(cards[index])  
 flipCardBack(element, getCardIndex(element, difficultyLevel))  
 }  
 }  
 btMedium.setOnClickListener{  
 restartTimer()  
 timerHandler.removeCallbacks(timerRunnable)  
 difficultyLevel = DifficultyLevel.MEDIUM  
 setGridLayout(difficultyLevel)  
 cardPairs = cardImagesMedium.size  
 cards.clear()  
 cards.addAll(cardImagesMedium)  
 cards.addAll(cardImagesMedium)  
 cards.shuffle()  
 imageViews = FindViews(this, difficultyLevel)  
 imageViews.forEach{element ->  
element.setOnClickListener(this)  
val index = getCardIndex(element, difficultyLevel)  
element.setImageResource(cards[index])  
flipCardBack(element, getCardIndex(element, difficultyLevel))  
 }  
 }  
 btHard.setOnClickListener{  
 restartTimer()  
 timerHandler.removeCallbacks(timerRunnable)  
 difficultyLevel = DifficultyLevel.HARD  
 setGridLayout(difficultyLevel)  
 cardPairs = cardImagesHard.size  
 cards.clear()  
 cards.addAll(cardImagesHard)  
 cards.addAll(cardImagesHard)  
 cards.shuffle()  
 imageViews = FindViews(this, difficultyLevel)  
 imageViews.forEach{element ->  
element.setOnClickListener(this)  
val index = getCardIndex(element, difficultyLevel)  
element.setImageResource(cards[index])  
flipCardBack(element, getCardIndex(element, difficultyLevel))  
 }  
 }  
 btTimer.setOnClickListener {  
 timerHandler.postDelayed(timerRunnable, 1000)  
 }  
 cardCount = when (difficultyLevel) {  
 DifficultyLevel.EASY -> 12  
 DifficultyLevel.MEDIUM -> 18  
 DifficultyLevel.HARD -> 24  
 }  
 }  
 override fun onClick(view: View) {  
val clickedCard = view as ImageView  
val cardIndex = getCardIndex(clickedCard, difficultyLevel)  
 if (isCardAlreadyFlipped(cardIndex, difficultyLevel)) {  
 return  
 }  
 flipCard(clickedCard, cardIndex)  
 if (firstCardIndex == -1) {  
 firstCardIndex = cardIndex  
 } else {  
 secondCardIndex = cardIndex  
 val firstCard = findViewById<ImageView>(getCardViewId(firstCardIndex,difficultyLevel ))  
 val secondCard = findViewById<ImageView>(getCardViewId(secondCardIndex, difficultyLevel))  
 if (cards[firstCardIndex] == cards[secondCardIndex]) {  
 flippedCardsCount += 2  
 checkGameCompletion()  
 } else {  
 val handler = Handler()  
 handler.postDelayed({  
 flipCardBack(firstCard, firstCardIndex)  
 flipCardBack(secondCard, secondCardIndex)  
 }, 1000)  
 }  
 firstCardIndex = -1  
 secondCardIndex = -1  
 }  
 }  
 private fun FindViews (context: Context, difficultyLevel: DifficultyLevel): List<ImageView> {  
 val cardList = mutableListOf<ImageView>()  
 when (difficultyLevel) {  
 DifficultyLevel.EASY -> {  
 // Обработчик щелчков для всех карточек

CardsAddEasy()   
 return (cardList)  
 }  
 DifficultyLevel.MEDIUM -> {

CardsAddMedium()   
 return (cardList)  
 }  
 DifficultyLevel.HARD -> {

CardsAddHard()   
 return (cardList)  
 }  
 }  
 }  
 private fun getCardIndex(card: ImageView, difficultyLevel: DifficultyLevel): Int {  
 when (difficultyLevel) {  
 DifficultyLevel.EASY -> {  
 return when (card.id) {  
 R.id.e\_card1 -> 0  
 R.id.e\_card2 -> 1  
 R.id.e\_card3 -> 2  
 R.id.e\_card4 -> 3  
 R.id.e\_card5 -> 4  
 R.id.e\_card6 -> 5  
 R.id.e\_card7 -> 6  
 R.id.e\_card8 -> 7  
 R.id.e\_card9 -> 8  
 R.id.e\_card10 -> 9  
 R.id.e\_card11 -> 10  
 R.id.e\_card12 -> 11  
 else -> -1  
 }  
 }  
 DifficultyLevel.MEDIUM -> {  
 return when (card.id) {  
 R.id.m\_card1 -> 0  
 R.id.m\_card2 -> 1  
 R.id.m\_card3 -> 2  
 R.id.m\_card4 -> 3  
 R.id.m\_card5 -> 4  
 R.id.m\_card6 -> 5  
 R.id.m\_card7 -> 6  
 R.id.m\_card8 -> 7  
 R.id.m\_card9 -> 8  
 R.id.m\_card10 -> 9  
 R.id.m\_card11 -> 10  
 R.id.m\_card12 -> 11  
 R.id.m\_card13 -> 12  
 R.id.m\_card14 -> 13  
 R.id.m\_card15 -> 14  
 R.id.m\_card16 -> 15  
 R.id.m\_card17 -> 16  
 R.id.m\_card18 -> 17  
 else -> -1  
 }  
 }  
 DifficultyLevel.HARD -> {  
 return when (card.id) {  
 R.id.h\_card1 -> 0  
 R.id.h\_card2 -> 1  
 R.id.h\_card3 -> 2  
 R.id.h\_card4 -> 3  
 R.id.h\_card5 -> 4  
 R.id.h\_card6 -> 5  
 R.id.h\_card7 -> 6  
 R.id.h\_card8 -> 7  
 R.id.h\_card9 -> 8  
 R.id.h\_card10 -> 9  
 R.id.h\_card11 -> 10  
 R.id.h\_card12 -> 11  
 R.id.h\_card13 -> 12  
 R.id.h\_card14 -> 13  
 R.id.h\_card15 -> 14  
 R.id.h\_card16 -> 15  
 R.id.h\_card17 -> 16  
 R.id.h\_card18 -> 17  
 R.id.h\_card19 -> 18  
 R.id.h\_card20 -> 19  
 R.id.h\_card21 -> 20  
 R.id.h\_card22 -> 21  
 R.id.h\_card23 -> 22  
 R.id.h\_card24 -> 23  
 else -> -1  
 }  
 }  
 }  
 }  
 private fun getCardViewId(cardIndex: Int, difficultyLevel: DifficultyLevel): Int {  
 when (difficultyLevel) {  
 DifficultyLevel.EASY -> {  
 return when (cardIndex) {  
 0 -> R.id.e\_card1  
 1 -> R.id.e\_card2  
 2 -> R.id.e\_card3  
 3 -> R.id.e\_card4  
 4 -> R.id.e\_card5  
 5 -> R.id.e\_card6  
 6 -> R.id.e\_card7  
 7 -> R.id.e\_card8  
 8 -> R.id.e\_card9  
 9 -> R.id.e\_card10  
 10 -> R.id.e\_card11  
 11 -> R.id.e\_card12  
 else -> -1  
 }  
 }  
 DifficultyLevel.MEDIUM -> {  
 return when (cardIndex) {  
 0 -> R.id.m\_card1  
 1 -> R.id.m\_card2  
 2 -> R.id.m\_card3  
 3 -> R.id.m\_card4  
 4 -> R.id.m\_card5  
 5 -> R.id.m\_card6  
 6 -> R.id.m\_card7  
 7 -> R.id.m\_card8  
 8 -> R.id.m\_card9  
 9 -> R.id.m\_card10  
 10 -> R.id.m\_card11  
 11 -> R.id.m\_card12  
 12 -> R.id.m\_card13  
 13 -> R.id.m\_card14  
 14 -> R.id.m\_card15  
 15 -> R.id.m\_card16  
 16 -> R.id.m\_card17  
 17 -> R.id.m\_card18  
 else -> -1  
 }  
 }  
 DifficultyLevel.HARD -> {  
 return when (cardIndex) {  
 0 -> R.id.h\_card1  
 1 -> R.id.h\_card2  
 2 -> R.id.h\_card3  
 3 -> R.id.h\_card4  
 4 -> R.id.h\_card5  
 5 -> R.id.h\_card6  
 6 -> R.id.h\_card7  
 7 -> R.id.h\_card8  
 8 -> R.id.h\_card9  
 9 -> R.id.h\_card10  
 10 -> R.id.h\_card11  
 11 -> R.id.h\_card12  
 12 -> R.id.h\_card13  
 13 -> R.id.h\_card14  
 14 -> R.id.h\_card15  
 15 -> R.id.h\_card16  
 16 -> R.id.h\_card17  
 17 -> R.id.h\_card18  
 18 -> R.id.h\_card19  
 19 -> R.id.h\_card20  
 20 -> R.id.h\_card21  
 21 -> R.id.h\_card22  
 22 -> R.id.h\_card23  
 23 -> R.id.h\_card24  
 else -> -1  
 }  
 }  
 }  
 }  
 private fun setGridLayout(difficultyLevel: DifficultyLevel) {  
 when (difficultyLevel) {  
 DifficultyLevel.EASY -> {  
 gridLayoutEasy.visibility = View.VISIBLE  
 gridLayoutMedium.visibility = View.GONE  
 gridLayoutHard.visibility = View.GONE  
 }  
 DifficultyLevel.MEDIUM -> {  
 gridLayoutEasy.visibility = View.GONE  
 gridLayoutMedium.visibility = View.VISIBLE  
 gridLayoutHard.visibility = View.GONE  
 }  
 DifficultyLevel.HARD -> {  
 gridLayoutEasy.visibility = View.GONE  
 gridLayoutMedium.visibility = View.GONE  
 gridLayoutHard.visibility = View.VISIBLE  
 }  
 }  
 }  
 private fun isCardAlreadyFlipped(cardIndex: Int, difficultyLevel: DifficultyLevel): Boolean {  
 val card = findViewById<ImageView>(getCardViewId(cardIndex, difficultyLevel))  
 return card.tag != null && card.tag == "flipped"  
 }  
 private fun flipCard(card: ImageView, cardIndex: Int) {  
 card.setImageResource(cards[cardIndex])  
 card.tag = "flipped"  
 }  
 private fun flipCardBack(card: ImageView, cardIndex: Int) {  
 card.setImageResource(R.drawable.card\_back)  
 card.tag = null  
 }  
 private fun checkGameCompletion() {  
 if (flippedCardsCount == cardPairs \* 2) {  
 timerHandler.removeCallbacks(timerRunnable)  
 val result\_old = timerTextView.text.toString()  
 val result = result\_old.replace("Время: ", "")  
 Toast.makeText(this, "Ваш результат: $result", Toast.LENGTH\_SHORT).show()  
 firebaseHelper.getUserStat { userStat ->  
 if (userStat != null) {  
 val updatedUserStat = when (difficultyLevel) {  
 DifficultyLevel.EASY -> {  
 if (userStat.best\_play1 != null) {  
 if (userStat.best\_play1!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_play1 = result)  
 }  
 } else {  
 userStat.copy(best\_play1 = result)  
 }  
 }  
 DifficultyLevel.MEDIUM -> {  
 if (userStat.best\_play2 != null) {  
 if (userStat.best\_play2!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_play2 = result)  
 }  
 } else {  
 userStat.copy(best\_play2 = result)  
 }  
 }  
 DifficultyLevel.HARD -> {  
 if (userStat.best\_play3 != null) {  
 if (userStat.best\_play3!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_play3 = result)  
 }  
 } else {  
 userStat.copy(best\_play3 = result)  
 }  
 }  
 else -> userStat  
 }  
 firebaseHelper.addStat(updatedUserStat) { success ->  
 if (success) {  
 // Обработка успешного сохранения статистики  
 } else {  
 // Обработка ошибки при сохранении статистики  
 }  
 }  
 }  
 else{  
 var newUserStat = UserStat("")  
 newUserStat = when (difficultyLevel) {  
DifficultyLevel.EASY -> newUserStat.copy(best\_play1 = result)  
DifficultyLevel.MEDIUM -> newUserStat.copy(best\_play2 = result)  
DifficultyLevel.HARD -> newUserStat.copy(best\_play3 = result)  
 else -> newUserStat  
 }  
 firebaseHelper.addStat(newUserStat) { success ->  
 if (success) {  
 // Обработка успешного сохранения статистики  
 } else {  
 // Обработка ошибки при сохранении статистики  
 }  
 }  
 }  
 }  
 }  
 }  
 private fun restartTimer() {  
 elapsedTime = 0  
 timerHandler.removeCallbacks(timerRunnable)  
 timerHandler.postDelayed(timerRunnable, 1000)  
 }  
 // указание элементов меню  
 override fun onCreateOptionsMenu(menu: Menu): Boolean {  
 menuInflater.inflate(R.menu.my\_menu, menu)  
 return true  
 }  
 override fun onPrepareOptionsMenu(menu: Menu): Boolean {  
 val deleteItem = menu.findItem(R.id.delete)  
 deleteItem.isVisible = false  
 return super.onPrepareOptionsMenu(menu)  
 }  
 fun updateOptionsMenu() {  
 invalidateOptionsMenu()  
 }  
 public fun onBackClick(item: MenuItem){  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 updateOptionsMenu()  
 }  
 // добавляем обработчик клика на кнопку удаления выбранных карточек  
 public fun onDeleteSelectedCardsButtonClick(item: MenuItem) { }  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 return when (item.itemId) {  
 R.id.back -> {  
 // Обработка нажатия кнопки "Назад"  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 true  
 }  
 else -> super.onOptionsItemSelected(item)  
 }  
 }  
}

**ShulteActivity.kt**

package com.example.hardbrain  
import …  
class ShulteActivity : AppCompatActivity() {  
 private lateinit var timerText: TextView  
 private lateinit var numberText: TextView  
 private lateinit var buttonGrid: Array<Array<Button>>  
 private lateinit var startTime: Date  
 private var currentNumber: Int = 1  
 private var gameStarted: Boolean = false  
 private var gameMode: Int = 3 // 1 - Классическая таблица, 2 - Буквенная таблица, 3 - Таблица с перемешиванием  
 private var handler: Handler = Handler()  
 private var updateTimeRunnable: Runnable? = null  
 private lateinit var firebaseHelper: FirebaseHelper  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_shulte)  
 timerText = findViewById(R.id.timerText)  
 numberText = findViewById(R.id.numberText)  
 firebaseHelper = FirebaseHelper()  
 val startButton: Button = findViewById(R.id.startButton)  
 startButton.setOnClickListener {  
 startGame()  
 }  
 val modeButton: Button = findViewById(R.id.modeButton)  
 modeButton.setOnClickListener {  
 showNextMode()  
 setNumberText(currentNumber)  
 }  
 buttonGrid = Array(5) { row ->  
 Array(5) { column ->  
 val buttonId = resources.getIdentifier("button$row$column", "id", packageName)  
 findViewById<Button>(buttonId).apply {  
 setOnClickListener {  
 onButtonClicked(row, column)  
 }  
 }  
 }  
 }  
 setGameModeText()  
 showNextMode()  
 setTimerText(0)  
 }  
 private fun startGame() {  
 if (gameStarted) return  
 gameStarted = true  
 currentNumber = 1  
 setNumberText(currentNumber)  
 shuffleButtons()  
 startTime = Date()  
 updateTimeRunnable = object : Runnable {  
 override fun run() {  
 val elapsedTime = Date().time - startTime.time  
 setTimerText(elapsedTime)  
 if (currentNumber > 25) {  
 gameStarted = false  
 handler.removeCallbacks(this)  
 showGameFinished()  
 } else {  
 handler.postDelayed(this, 1000)  
 }  
 }  
 }  
 handler.postDelayed(updateTimeRunnable as Runnable, 1000) }  
 private fun shuffleButtons() {  
 val numbers = mutableListOf<Int>()  
 when (gameMode) {  
 1 -> numbers.addAll(1..25)  
 2 -> numbers.addAll('A'.code..'Y'.code)  
 3 -> { numbers.addAll(1..25)  
 }  
 }  
 numbers.shuffle()  
 for (row in 0 until 5) {  
 for (column in 0 until 5) {  
 val number = numbers[row \* 5 + column]  
 val button = buttonGrid[row][column]  
 button.text = getButtonText(number)  
 button.isEnabled = true  
 }  
 }  
 }  
 private fun shuffleEnabledButtons() {  
 val enabledButtons = buttonGrid.flatten().filter { it.isEnabled }  
 val enabledButtonTexts = enabledButtons.map { it.text.toString() } as MutableList  
 enabledButtonTexts.shuffle()  
 for ((index, button) in enabledButtons.withIndex()) {  
 button.text = enabledButtonTexts[index]  
 }  
 }  
 private fun onButtonClicked(row: Int, column: Int) {  
 val button = buttonGrid[row][column]  
 val buttonNumber = getButtonNumber(button.text.toString())  
 if (buttonNumber == currentNumber) {  
 button.isEnabled = false  
 currentNumber++  
 if (currentNumber > 25) {  
 gameStarted = false  
 handler.removeCallbacksAndMessages(null)  
 showGameFinished()  
 } else {  
 if (gameMode == 3) {shuffleEnabledButtons()}  
 setNumberText(currentNumber)  
 }  
 }  
 }  
 private fun showNextMode() {  
 gameMode = (gameMode % 3) + 1  
 setGameModeText()  
 resetGame()  
 }  
 private fun setGameModeText() {  
 val modeText: TextView = findViewById(R.id.modeText)  
 val modeName = when (gameMode) {  
 1 -> "Классическая таблица"  
 2 -> "Буквенная таблица"  
 3 -> "Таблица с перемешиванием"  
 else -> ""  
 }  
 modeText.text = "Режим: $modeName"  
 }  
 private fun setTimerText(elapsedTime: Long) {  
 val seconds = elapsedTime / 1000  
 timerText.text = "Время: $seconds сек"  
 }  
 private fun setNumberText(number: Int) {  
 val num = (number + 'A'.code - 1).toChar().toString()  
 return when (gameMode) {  
 1 -> numberText.text = "Текущее число: $number"  
 2 -> numberText.text = "Текущий символ: $num"  
 3 -> numberText.text = "Текущее число: $number"  
 else -> numberText.text = ""  
 }  
 }  
 private fun showGameFinished() {  
 val result\_old = timerText.text.toString()  
 val result = result\_old.replace("Время: ", "")  
 Toast.makeText(this, "Ваш результат: $result", Toast.LENGTH\_SHORT).show()  
 firebaseHelper.getUserStat { userStat ->  
 if (userStat != null) {  
 val updatedUserStat = when (gameMode) {  
 1 -> {  
 if (userStat.best\_shulte1 != null) {  
 if (userStat.best\_shulte1!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_shulte1 = result)  
 }  
 } else {  
 userStat.copy(best\_shulte1 = result)  
 }  
 }  
 2 -> {  
 if (userStat.best\_shulte2 != null) {  
 if (userStat.best\_shulte2!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_shulte2 = result)  
 }  
 } else {  
 userStat.copy(best\_shulte2 = result)  
 }  
 }  
 3 -> {  
 if (userStat.best\_shulte3 != null) {  
 if (userStat.best\_shulte3!!.split(" ")[0].toInt() < result.split(" ")[0].toInt()) {  
 userStat.copy()  
 }  
 else{  
 userStat.copy(best\_shulte3 = result)  
 }  
 } else {  
 userStat.copy(best\_shulte3 = result)  
 }  
 }  
 else -> userStat  
 }  
 firebaseHelper.addStat(updatedUserStat) { success ->  
 if (success) {  
 // Обработка успешного сохранения статистики  
 } else {  
 // Обработка ошибки при сохранении статистики  
 }  
 }  
 }  
 else{  
 var newUserStat = UserStat("")  
 newUserStat = when (gameMode) {  
 1 -> newUserStat.copy(best\_shulte1 = result)  
 2 -> newUserStat.copy(best\_shulte2 = result)  
 3 -> newUserStat.copy(best\_shulte3 = result)  
 else -> newUserStat  
 }  
 firebaseHelper.addStat(newUserStat) { success ->  
 if (success) {  
 // Обработка успешного сохранения статистики  
 } else {  
 // Обработка ошибки при сохранении статистики  
 }  
 }  
 }  
 }  
 }  
 private fun getButtonText(number: Int): String {  
 return when (gameMode) {  
 1 -> number.toString()  
 2 -> number.toChar().toString()  
 3 -> number.toString()  
 else -> number.toString()  
 }  
 }  
 private fun getButtonNumber(text: String): Int {  
 return when (gameMode) {  
 1 -> text.toInt()  
 2 -> text[0].code - 'A'.code + 1  
 3 -> text.toInt()  
 else -> 0  
 }  
 }  
 private fun resetGame() {  
 currentNumber = 1  
 shuffleButtons()  
 setTimerText(0)  
 setNumberText(1)  
 // Остановка и сброс таймера  
 if (gameStarted) {  
 updateTimeRunnable?.let { handler.removeCallbacks(it) }  
 gameStarted = false  
 }  
 }  
 override fun onDestroy() {  
 super.onDestroy()  
 handler.removeCallbacksAndMessages(null)  
 }  
 // указание элементов меню  
 override fun onCreateOptionsMenu(menu: Menu): Boolean {  
 menuInflater.inflate(R.menu.my\_menu, menu)  
 return true  
 }  
 override fun onPrepareOptionsMenu(menu: Menu): Boolean {  
 val deleteItem = menu.findItem(R.id.delete)  
 deleteItem.isVisible = false  
 return super.onPrepareOptionsMenu(menu)  
 }  
 fun updateOptionsMenu() {  
 invalidateOptionsMenu()  
 }  
  
 public fun onBackClick(item: MenuItem){  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 updateOptionsMenu()  
 }  
 // добавляем обработчик клика на кнопку удаления выбранных карточек  
 public fun onDeleteSelectedCardsButtonClick(item: MenuItem) { }  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 return when (item.itemId) {  
 R.id.back -> {  
 // Обработка нажатия кнопки "Назад"  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 finish()  
 true  
 }  
 else -> super.onOptionsItemSelected(item)  
 }  
 }  
}

**ПРИЛОЖЕНИЕ Б**

Результаты функционального тестирования представлены в таблице Б.1.

Таблица Б.1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Действие | Ожидаемый результат | Результат |
| 1 | Первый вход в приложение | Переход на страницу авторизации | Пройдено |
| 2 | Авторизация в приложении с помощью Google аккаунта | Пройденная авторизация в приложении с последующим переходом на главную страницу | Пройдено |
| 3 | Добавление новой коллекции/  карточки | При добавлении всей необходимой информации коллекции/карточки добавляется в БД, иначе выводится сообщение о неполноте данных. Информация на странице обновляется | Пройдено |
| 4 | Изменение существующей коллекции/  карточки | При изменении любой из характеристик остальные данные сохраняются, изменения применяются и заносятся в БД, информация на странице обновляется | Пройдено |
| 5 | Удаление коллекций/ карточек | При выборе коллекций/карточек и их последующего удаления необходимая информация удаляется из БД, информация на странице обновляется | Пройдено |

Продолжение Таблицы Б.1

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Нажатие на кнопку «Поделиться» | При нажатии на соответствующую иконку коллекции она выгружается в общий список коллекций в БД и становится доступна для просмотра и добавления другими пользователями | Пройдено |
| 7 | Удаление коллекции из общего доступа | Возможность удаления коллекции появляется только в случае, если коллекция создана пользователем. При удалении коллекции данные удаляются из БД и она становится не доступна для других пользователей | Пройдено |
| 8 | Добавление коллекции из общего доступа | При нажатии на иконку добавления коллекции из общего доступа она появляется в активности личных коллекций и заносится в БД | Пройдено |
| 9 | Выбор коллекций для повторения карточек с помощью метода интервального повторения | Карточки из выбранной коллекции отображаются на соответствующей активности при соблюдении условий | Пройдено |
| 10 | Повторение «ежедневных» карточек | При оценке степени запоминания карточки дата ее следующего повторения изменяется в соответствии с формулой, аналогично изменяются фактор и интервал повторения | Пройдено |

Продолжение Таблицы Б.1

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | Переворачивание «ежедневной» карточки | При нажатии на лицевую сторону карточки, она переворачивается, содержимое лицевой стороны скрывается, становится видимым содержимое обратной стороны | Пройдено |
| 12 | Раскрытие карточек на активности карточек | При нажатии на карточку она «раскрывается» и становится видимым не только содержимое лицевой стороны, но и оборотной. При повторном нажатии содержимое обратной стороны скрывается, карточка «закрывается» | Пройдено |
| 13 | Отображение разных уровней игры «Найди пару» | При выборе уровня сложности выводится корректное содержимое активности | Пройдено |
| 14 | Составление пары | В случае составления пары обе карточки остаются перевернутыми, иначе они переворачиваются «рубашкой» вверх | Пройдено |
| 15 | Завершение игры | При составлении всех пар игра заканчивается, останавливается таймер и при выполнении необходимых условий результат записывается в статистику | Пройдено |
| 16 | Отображение разных уровней упражнения «таблица Шульте» | При выборе уровня сложности выводится корректное содержимое активности | Пройдено |

Продолжение Таблицы Б.1

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | Завершение упражнения «таблица Шульте» | При составлении всех пар игра заканчивается, останавливается таймер и при выполнении необходимых условий результат записывается в статистику | Пройдено |
| 18 | Вывод статистики | Статистические данные верны и выводятся корректно | Пройдено |
| 19 | Вывод данных пользователя | На странице пользователя данные отображаются корректно | Пройдено |

**ПРИЛОЖЕНИЕ В**

Скриншоты экрана приложения представлены на рисунках В.1-В.6

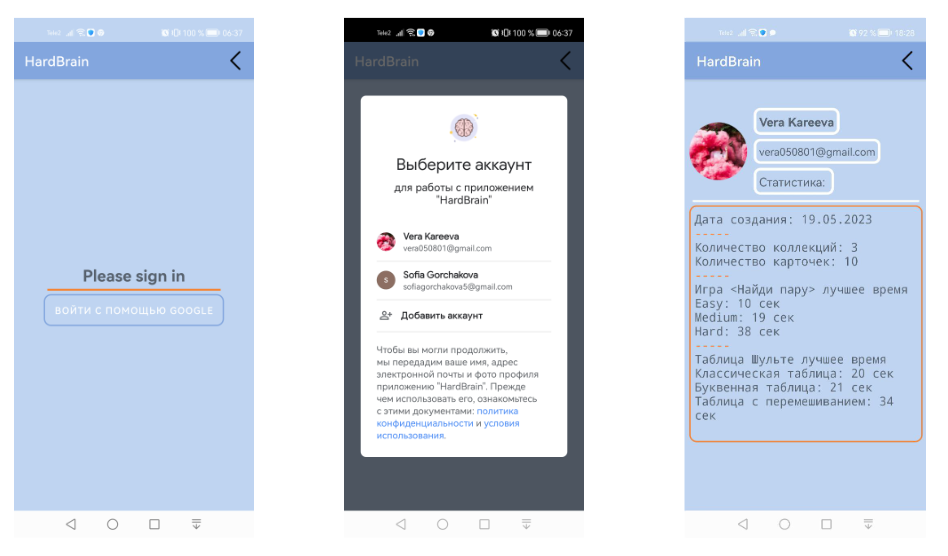


Рисунок В.1 – Экран входа и профиля

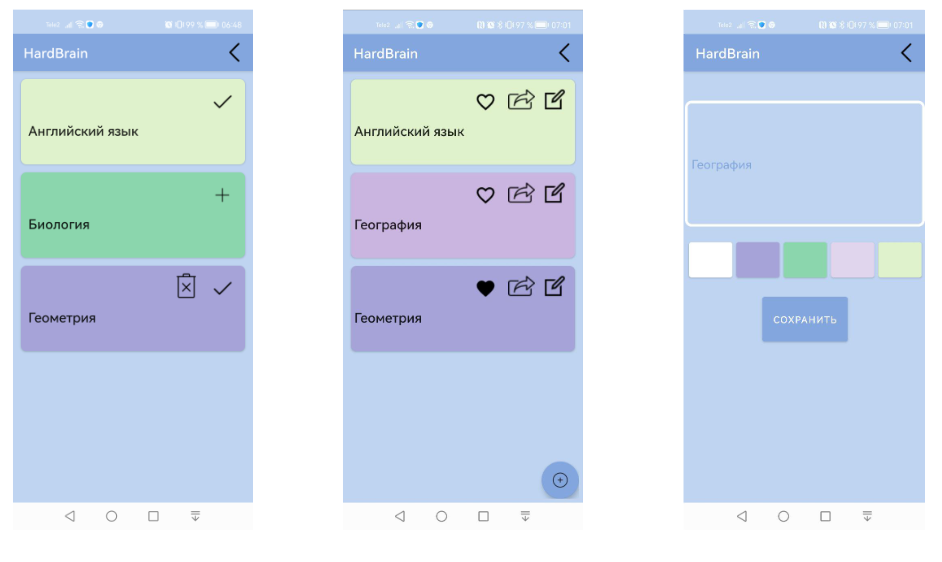


Рисунок В.2 – Экраны коллекций

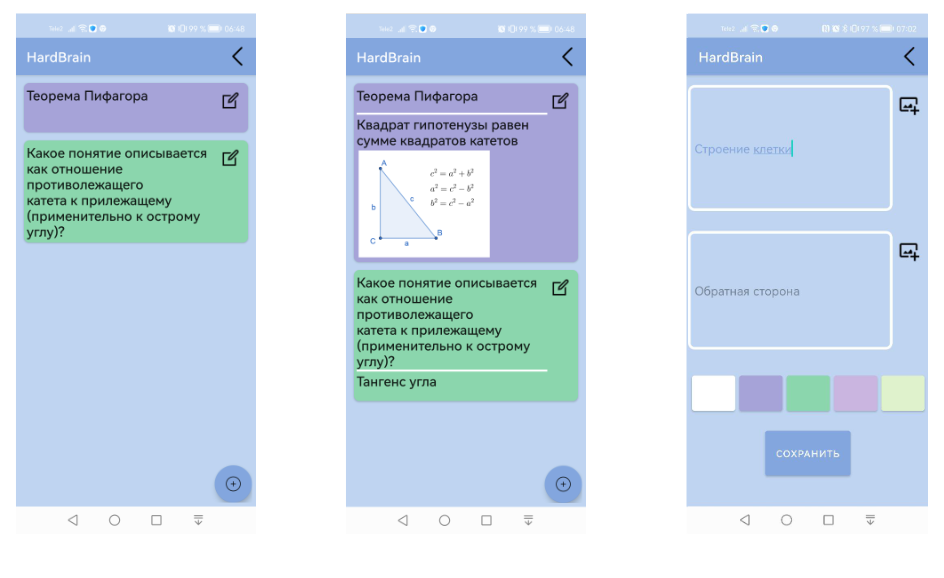


Рисунок В.3 – Экраны карточек



Рисунок В.4 – Экран с методом интервального повторения

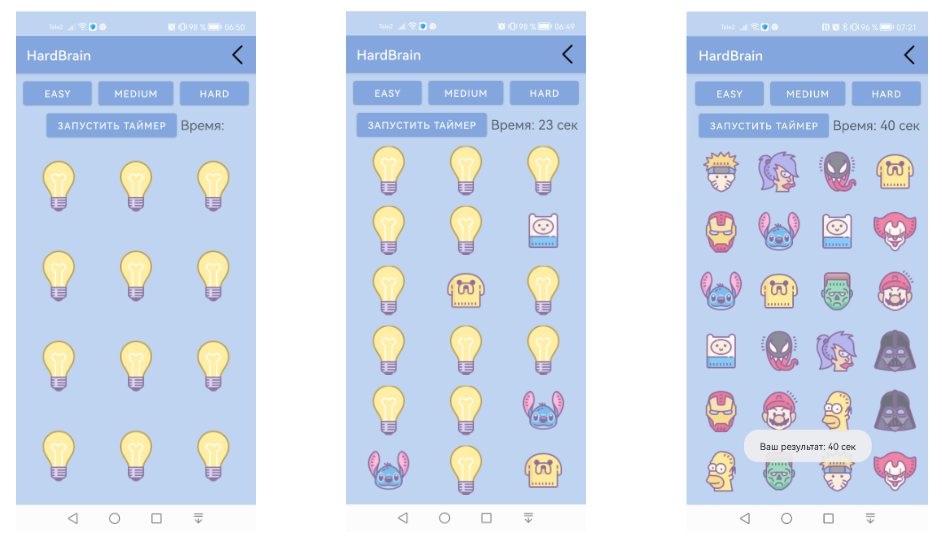


Рисунок В.5 – Экран с игрой «Найди пару»

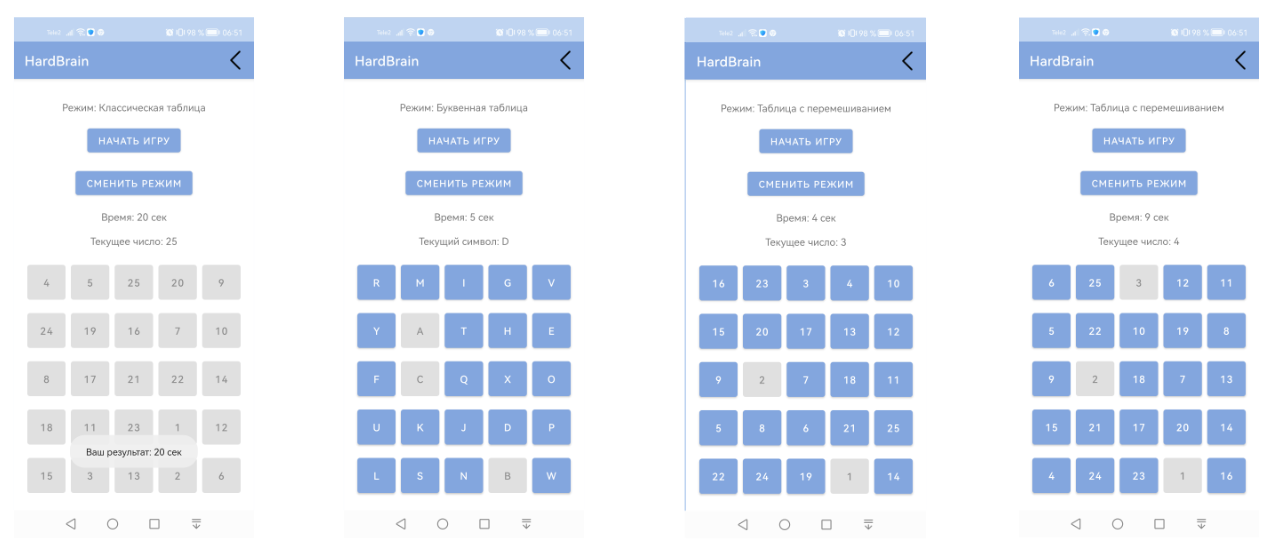


Рисунок В.6 – Экран с упражнением «таблица Шульте»