Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Уфимский государственный нефтяной технический университет

Кафедра вычислительной техники и инженерной кибернетики

КУРСОВАЯ РАБОТА
по дисциплине
Мобильные приложения и программирование устройств

ПРИЛОЖЕНИЕ ДЛЯ ИЗУЧЕНИЯ ТЕХНИК СКОРОЧТЕНИЯ

Выполнил студент группы БПО-22-01 А.Ф. Абдулбасырова
Принял ст. преподаватель Е.В. Дружинская

Дата представления работы: _____
Дата защиты: _____ Результат: _____

Уфа, 2025 г.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1. **Android** — мобильная операционная система, разработанная компанией Google, используемая в качестве целевой платформы для разработки приложения.
2. **API** — программный интерфейс приложения (Application Programming Interface), обеспечивающий взаимодействие между компонентами приложения.
3. **IDE** — интегрированная среда разработки (Integrated Development Environment), использованная для проектирования и программирования приложения (например, Android Studio).
4. **Kotlin** — язык программирования, применённый для реализации программной логики приложения.
5. **RSVP** — метод быстрого последовательного визуального представления текста (Rapid Serial Visual Presentation).
6. **UI** — пользовательский интерфейс (User Interface), включающий визуальные элементы приложения, такие как экраны и элементы управления.
7. **XML** — расширяемый язык разметки (Extensible Markup Language), используемый для создания макетов пользовательского интерфейса и хранения текстовых ресурсов.

ВВЕДЕНИЕ

В условиях стремительного роста информационных потоков способность быстро и эффективно воспринимать текстовую информацию становится ключевым навыком для людей любого возраста. Современные пользователи сталкиваются с перегрузкой графической информацией, что приводит к трудностям в чтении, обусловленным узким полем зрения, регрессией глаз, привычкой к артикуляции при чтении и недостаточной концентрацией внимания. Навык скорочтения позволяет ускорить обработку текстов, улучшить понимание прочитанного, развить память и повысить продуктивность. Разработка мобильного приложения для тренировки скорочтения представляет актуальную задачу, так как обеспечивает доступный и удобный инструмент для совершенствования навыков чтения. Использование платформы Android, доминирующей на рынке мобильных устройств, делает приложение широко доступным для пользователей.

Цель работы. Обосновать актуальность и значимость разработки мобильного приложения для тренировки скорочтения, а также спроектировать и реализовать программный продукт, направленный на повышение скорости и качества восприятия текстовой информации для широкой аудитории.

Задачи. Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить предметную область, включая особенности скорочтения и потребности пользователей в развитии этого навыка.
2. Провести анализ существующих программных продуктов для тренировки скорочтения, оценив их функциональность, дизайн и логику реализации.
3. Выполнить проектирование мобильного приложения, включающее функциональное моделирование, эскизирование пользовательского интерфейса и логическое моделирование.

4. Реализовать программную часть приложения, включая разработку логики и пользовательского интерфейса.

5. Провести тестирование приложения и сопоставить результаты с поставленной целью.

Объект исследования. Процесс разработки мобильных приложений для операционной системы Android, направленных на совершенствование навыков восприятия информации.

Предмет исследования. Мобильное приложение для тренировки скорочтения, включая его функциональные возможности, дизайн и программную реализацию.

Структура работы включает следующие разделы: обозначения и сокращения, введение, обзор предметной области, проектирование приложения, программная реализация, заключение и список использованных источников. Введение обосновывает актуальность темы, цель и задачи проекта. Раздел «Обзор предметной области» включает анализ особенностей скорочтения и существующих решений. Раздел «Проектирование приложения» охватывает функциональное моделирование, эскизирование интерфейса и логическое моделирование. В разделе «Программная реализация» описывается разработка и тестирование приложения. В заключении подводятся итоги работы и определяются перспективы дальнейшего развития приложения.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Изучение техник скорочтения

Скорочтение — это совокупность техник, направленных на увеличение скорости чтения при сохранении высокого уровня понимания и усвоения текстовой информации. В условиях информационного общества, где люди ежедневно сталкиваются с большими объёмами текстов, такие техники становятся важным инструментом для повышения эффективности обработки информации. Ниже приведён обзор ключевых техник скорочтения, которые помогают развивать навыки быстрого чтения, улучшать концентрацию и память:

1. **Чтение блоками.** Эта техника предполагает восприятие текста не по отдельным словам, а целыми группами слов или фразами. Чтение блоками позволяет расширить поле зрения, охватывая больше текста за один взгляд, и сократить время, затрачиваемое на движение глаз. Это уменьшает количество фиксаций глаз на странице и ускоряет процесс чтения.

2. **Чтение по диагонали.** Данная техника направлена на быстрое сканирование текста по диагонали для улавливания общей структуры и ключевых идей без детального чтения каждого слова. Читатель перемещает взгляд по диагональным траекториям, фокусируясь на основных элементах текста, что позволяет быстро понять суть материала.

3. **Поиск ключевых слов.** Техника заключается в выделении наиболее значимых слов, несущих основную смысловую нагрузку, с игнорированием второстепенных элементов, таких как предлоги, союзы или описательные слова. Это помогает сосредоточиться на главном содержании текста, ускоряя его восприятие и понимание.

4. **Метод указки.** Использование указки, например пальца, ручки или другого ориентира, помогает направлять взгляд вдоль строки текста, поддерживая постоянный ритм чтения. Эта техника минимизирует регрессию глаз (непроизвольное возвращение к уже прочитанным словам) и способствует более плавному и быстрому чтению.

5. **Слова наоборот.** Данная техника предполагает чтение слов в обратном порядке, что развивает гибкость восприятия текста и зрительное внимание. Практика чтения слов в обратной последовательности тренирует мозг быстрее обрабатывать текстовую информацию и адаптироваться к нестандартным форматам текста.

6. **Предложения наоборот.** Эта методика включает чтение предложений, написанных в обратном порядке, что помогает развивать память и способность анализировать структуру текста. Читатель учится восстанавливать смысл перевёрнутых предложений, что улучшает навыки реконструкции текста и понимания его логики.

Перечисленные техники скорочтения направлены на преодоление типичных препятствий, таких как узкое поле зрения, регрессия глаз, артикуляция (внутреннее проговаривание текста) и недостаточная концентрация. Их использование позволяет пользователям быстрее обрабатывать информацию, улучшать запоминание и повышать продуктивность в работе с текстами.

1.2 Анализ существующих решений

Для формирования концепции разрабатываемого приложения был проведён анализ существующих программных продуктов, предназначенных для тренировки скорочтения. Рассмотрены следующие приложения, доступные на платформе Android: Quickify, Spritz, Readmical и Spreeder. Анализ включает описание их функциональности, преимуществ и недостатков.

1 **Quickify**

Приложение предлагает различные упражнения для развития навыков скорочтения, включая визуальные тренажёры, слепое чтение, распознавание слов и букв, тренировки памяти и внимания. Интерфейс интуитивно понятный, есть встроенные рекомендации и статистика прогресса.

Преимущества: разнообразие упражнений, наглядная визуализация результатов, мотивационные элементы.

Недостатки: часть функций доступна только в платной версии, перегруженность интерфейса в отдельных модулях.

2 **Readmical**

Простое в использовании приложение, основное внимание в котором уделено технике чтения с помощью RSVP (Rapid Serial Visual Presentation) — последовательному отображению слов по одному на экране. Поддерживается загрузка собственных текстов.

Преимущества: минималистичный интерфейс, быстрая работа, возможность кастомизации скорости.

Недостатки: ограниченная функциональность, отсутствуют комплексные тренировки памяти и внимания.

3 **Spritz**

Это приложение реализует фирменную технологию Spritz для показа текста с фиксацией ключевой буквы (Optimal Recognition Point), что позволяет сократить время на перемещение взгляда.

Преимущества: инновационная подача текста, высокая скорость восприятия при адаптации, компактный и удобный интерфейс.

Недостатки: ограниченные настройки, сложности в восприятии длинных и сложных предложений, особенно при высоких скоростях.

4 **Spreeder**

Многофункциональное приложение, предназначенное не только для тренировки скорочтения, но и для улучшения концентрации, памяти и когнитивных способностей. Имеет интеграцию с облачными сервисами и синхронизацию между устройствами.

Преимущества: широкие возможности персонализации, наличие обучающих программ и заданий, профессиональный подход.

Недостатки: англоязычный интерфейс, высокая цена подписки, перегруженность функционалом для неподготовленного пользователя.

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

2.1 Функциональная модель

Функциональное моделирование определяет сценарии использования приложения и его основные возможности, обеспечивая соответствие потребностям пользователей. Приложение предназначено для тренировки скорочтения с использованием техник, таких как чтение блоками, чтение по диагонали, поиск ключевых слов, метод указки, слова наоборот и предложения наоборот.

Основные сценарии использования включают:

— **Выбор техники скорочтения**: Пользователь выбирает одну из доступных техник для тренировки (например, чтение блоками или метод указки).

— **Настройка параметров тренировки**: Пользователь задаёт уровень скорости отображения текста.

— **Выполнение упражнений**: Пользователь проходит тренировочные задания, соответствующие выбранной технике, с отображением текста в заданном формате (например, по диагонали или в обратном порядке).

— **Просмотр статистики**: Пользователь получает доступ к результатам тренировок, включая скорость чтения, количество правильных ответов.

Таким образом, функциональная модель приложения охватывает весь цикл взаимодействия пользователя — от выбора техники и настройки параметров до выполнения упражнений и анализа результатов. Такой подход обеспечивает индивидуализацию процесса обучения, способствует системному развитию навыков скорочтения и позволяет пользователю отслеживать личный прогресс, адаптируя тренировки под свои цели и уровень подготовки.

2.2 Эскизирование экранов

Эскизирование экранов является ключевым этапом проектирования пользовательского интерфейса мобильного приложения для тренировки

скорочтения. Цель этого этапа — разработать интуитивно понятный, функциональный и визуально привлекательный интерфейс, который обеспечит удобное взаимодействие пользователя с приложением и поддержит выполнение всех сценариев использования, описанных в функциональной модели. На основе анализа потребностей пользователей и функциональных требований были спроектированы основные экраны приложения, каждый из которых соответствует определённому этапу взаимодействия: выбор техники, настройка параметров, выполнение упражнений, просмотр результатов и доступ к дополнительным материалам.

Эскизирование проводилось в среде Android Studio с использованием XML-макетов, что позволило одновременно проектировать визуальную структуру и интегрировать её с программной логикой. Макеты создавались с акцентом на простоту навигации, минимализм и поддержку всех описанных техник скорочтения: чтение блоками, чтение по диагонали, поиск ключевых слов, метод указки, слова наоборот и предложения наоборот. Для демонстрации интерфейса были подготовлены скриншоты экранов, которые иллюстрируют их структуру и функциональность (рисунки 1 - 8). Ниже описаны основные экраны приложения, их назначение и ключевые элементы интерфейса, реализованные в XML.

Основные экраны приложения

1. Экран главного меню.

Назначение: Обеспечивает навигацию по разделам приложения.

Элементы интерфейса: Нижняя панель навигации с вкладками «Упражнения», «Рейтинг», «Материалы».

XML-макет: activity_main.xml.

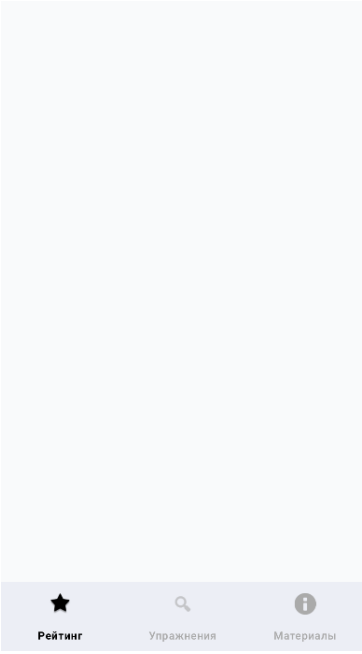

Рис. 1 – Экран главного меню

2. Экран выбора техники.

Назначение: Позволяет выбрать технику скорочтения.

Элементы интерфейса: Список техник (RecyclerView), иконка справки, открывающая подсказку.

XML-макет: fragment_exercises.xml.

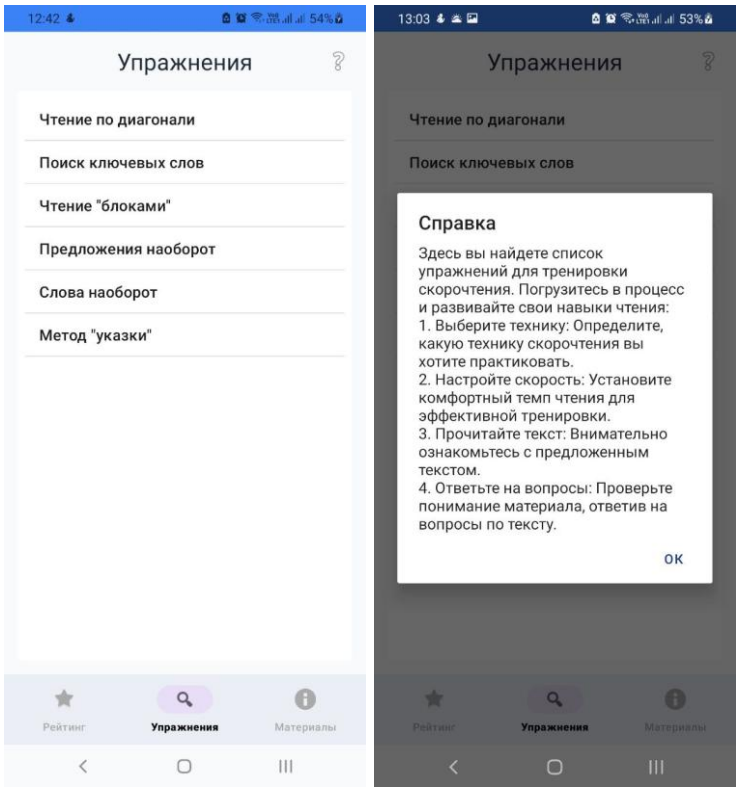

Рис. 2-3 – Экран выбора техники и открывающаяся подсказка

3. Экран выбора скорости.

Назначение: Настройка скорости чтения перед тренировкой.

Элементы интерфейса: Заголовок техники, кнопки для выбора скорости (200, 400, 600 слов/мин).

XML-макет: fragment_speed_selection.xml.

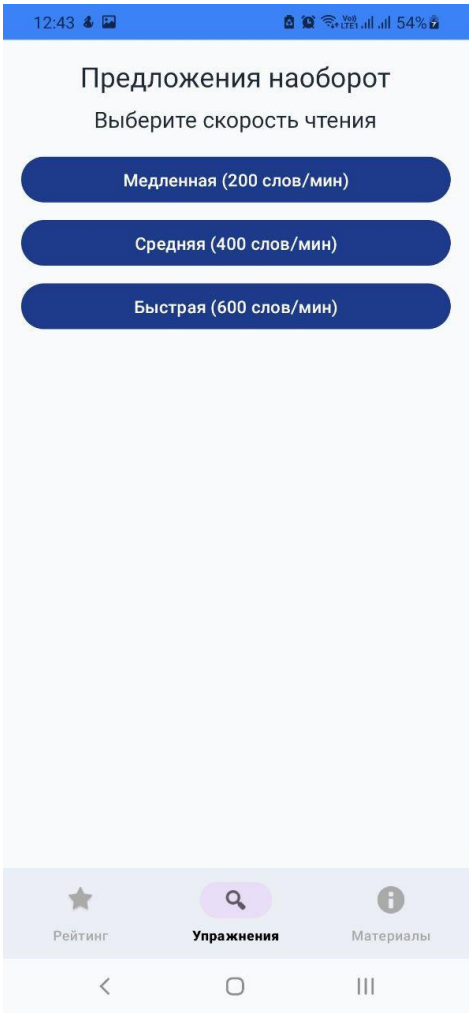

Рис. 4 – Экран выбора скорости

4. Экран выполнения упражнения.

Назначение: Отображает текст с применением выбранной техники.

Элементы интерфейса: Текстовая область (TextView) для отображения текста.

XML-макет: fragment_reading_test.xml.

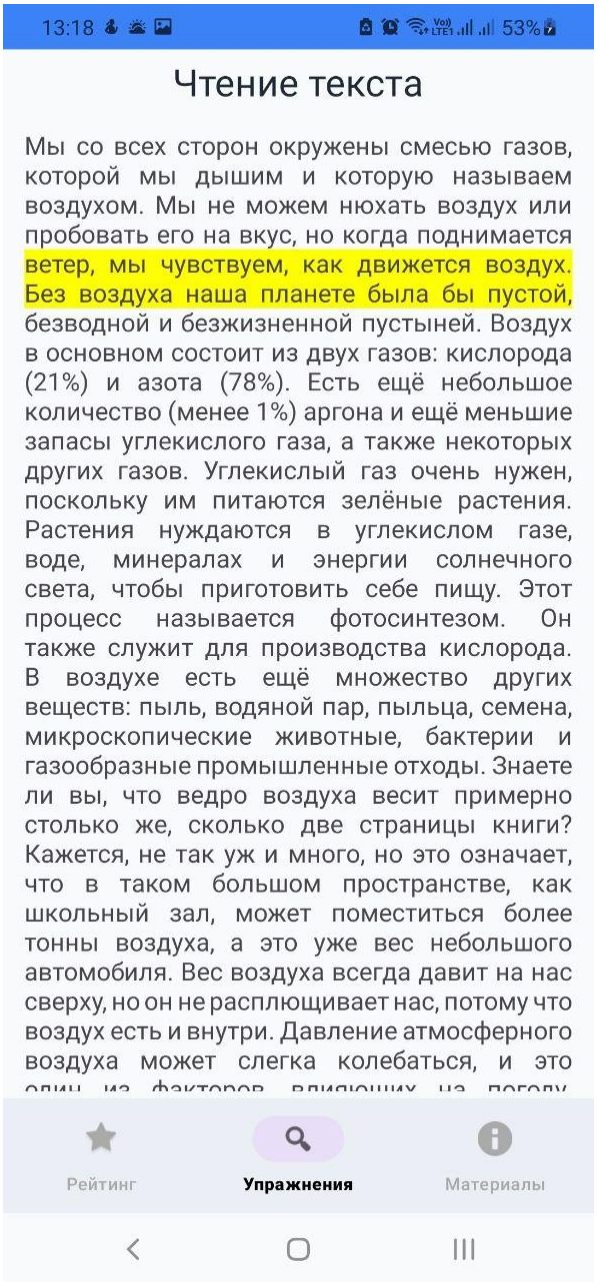

Рис. 5 – Экран выполнения упражнения

5. Экран тестирования.

Назначение: Проверяет понимание текста через вопросы.

Элементы интерфейса: Заголовок вопроса, текст вопроса, варианты ответа (RadioGroup), кнопка «Отправить».

XML-макет: fragment_test.xml.

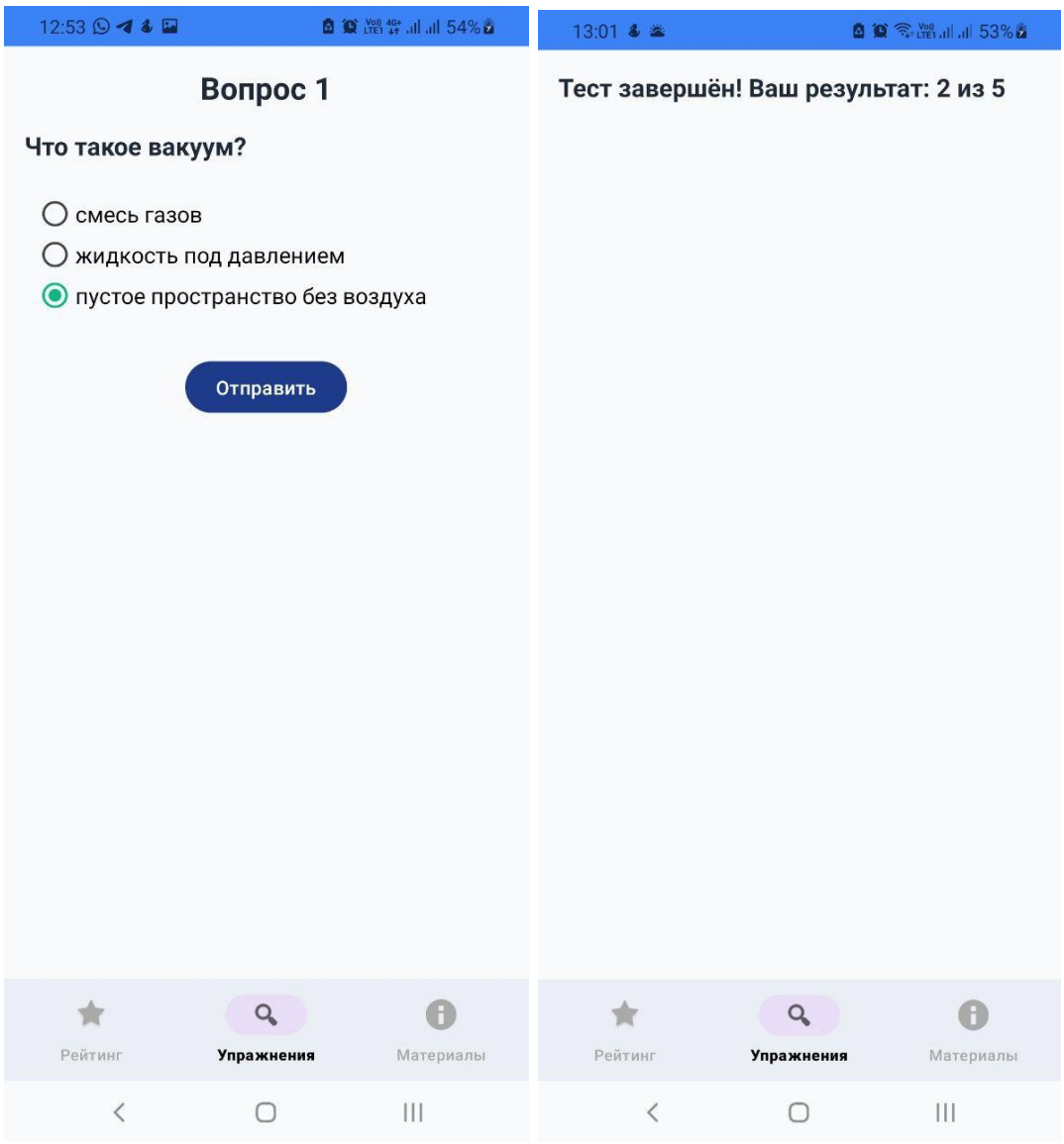

Рис. 6 – Экран тестирования и выведенного резуталта

6. Экран рейтинга.

Назначение: Показывает лучшие результаты по техникам.

Элементы интерфейса: Список результатов (RecyclerView), иконка справки с подсказкой.

XML-макет: fragment_rating.xml.

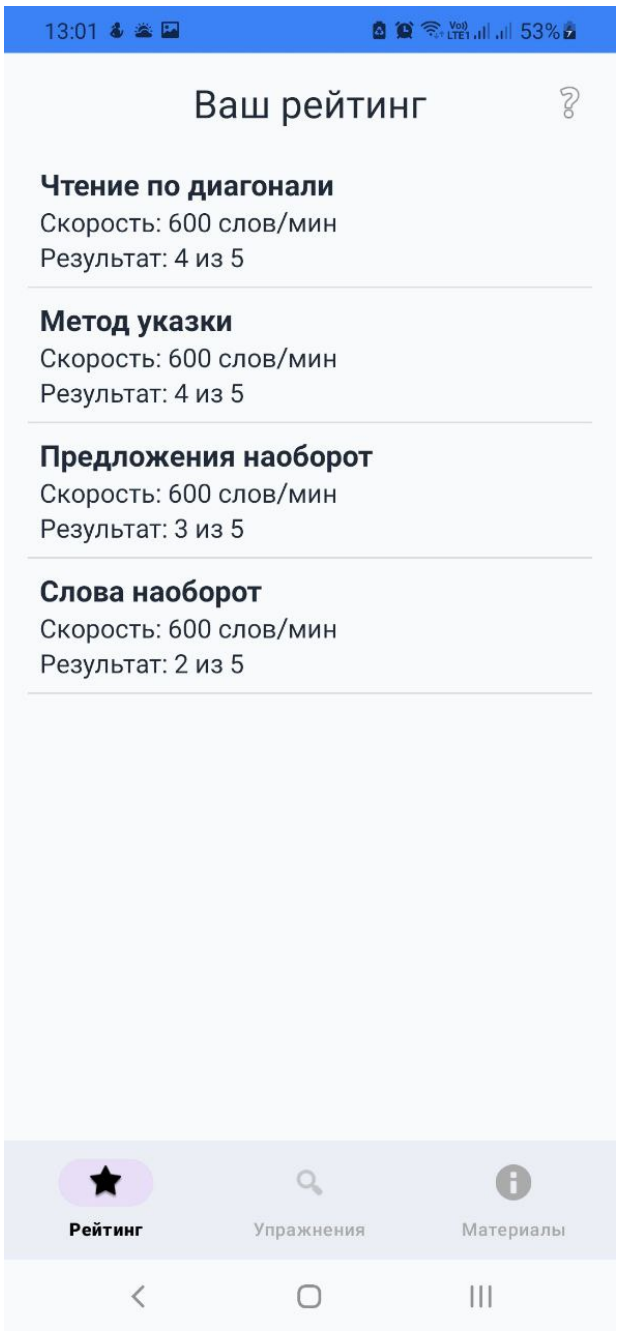

Рис. 7 – Экран рейтинга

7. Экран дополнительных материалов.

Назначение: Предоставляет информацию о техниках.

Элементы интерфейса: Список техник (RecyclerView), иконка справки с подсказкой.

XML-макет: fragment_materials.xml.

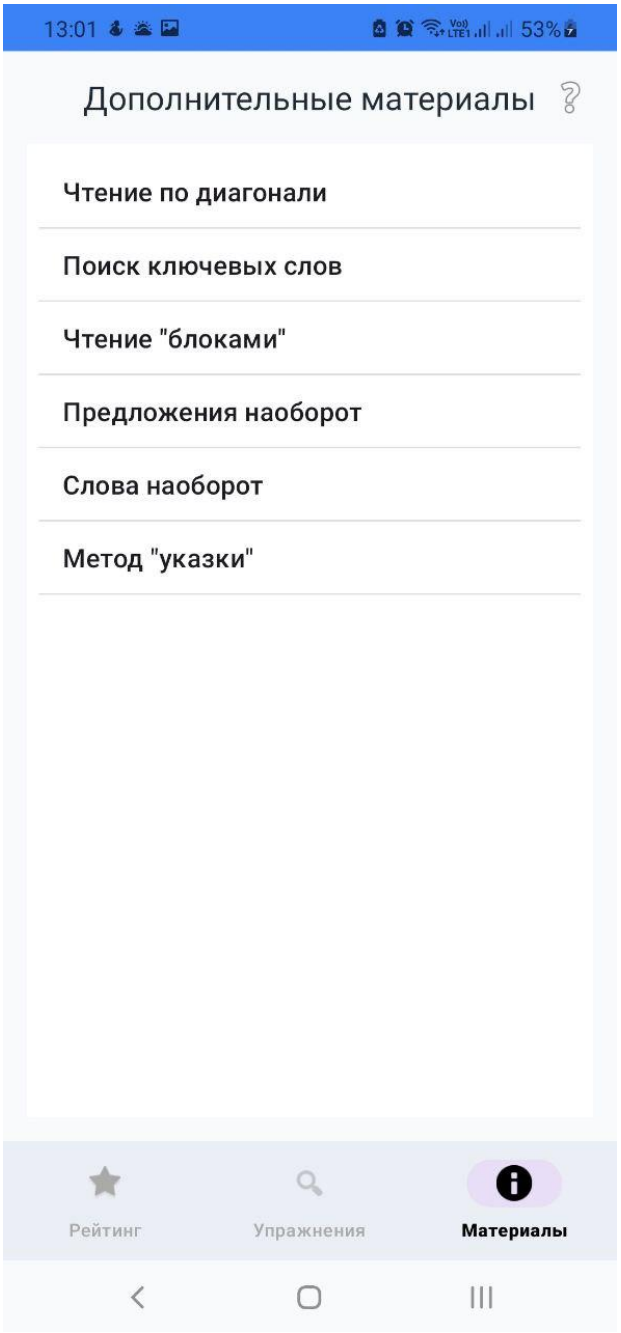

Рис. 8 – Экран дополнительных материалов

8. Экран описания техники.

Назначение: Показывает детали техники и запускает тренировку.

Элементы интерфейса: Кнопка «<» (Назад), описание техники, кнопка «Старт», контейнеры для предварительного просмотра.

XML-макет: fragment_technique_detail.xml.

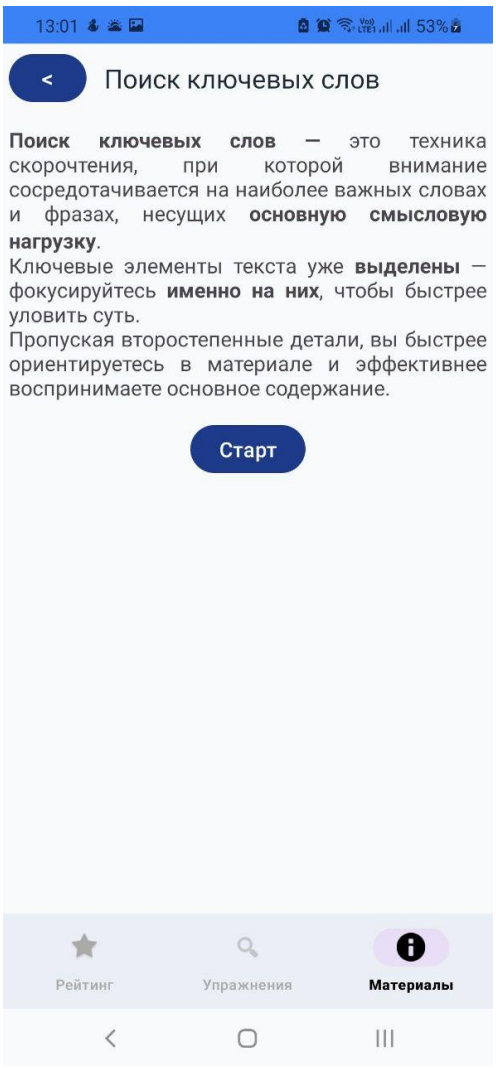

Рис. 9 – Экран описания техники

Принципы дизайна

- **Удобство использования**: Элементы управления (кнопки, списки) и иконки справки размещены интуитивно, обеспечивая лёгкий доступ к функциям.
- **Минимализм**: Экраны содержат только необходимые элементы, упрощая навигацию и фокусируя внимание на задаче.
- **Целевая функциональность**: Каждый экран чётко реализует свой сценарий: выбор техники, тренировка, тестирование или просмотр результатов.
- **Динамичность**: Подсказки и анимации (например, для метода указки или диагонального чтения) делают взаимодействие более живым и понятным.

2.3 Логическая модель

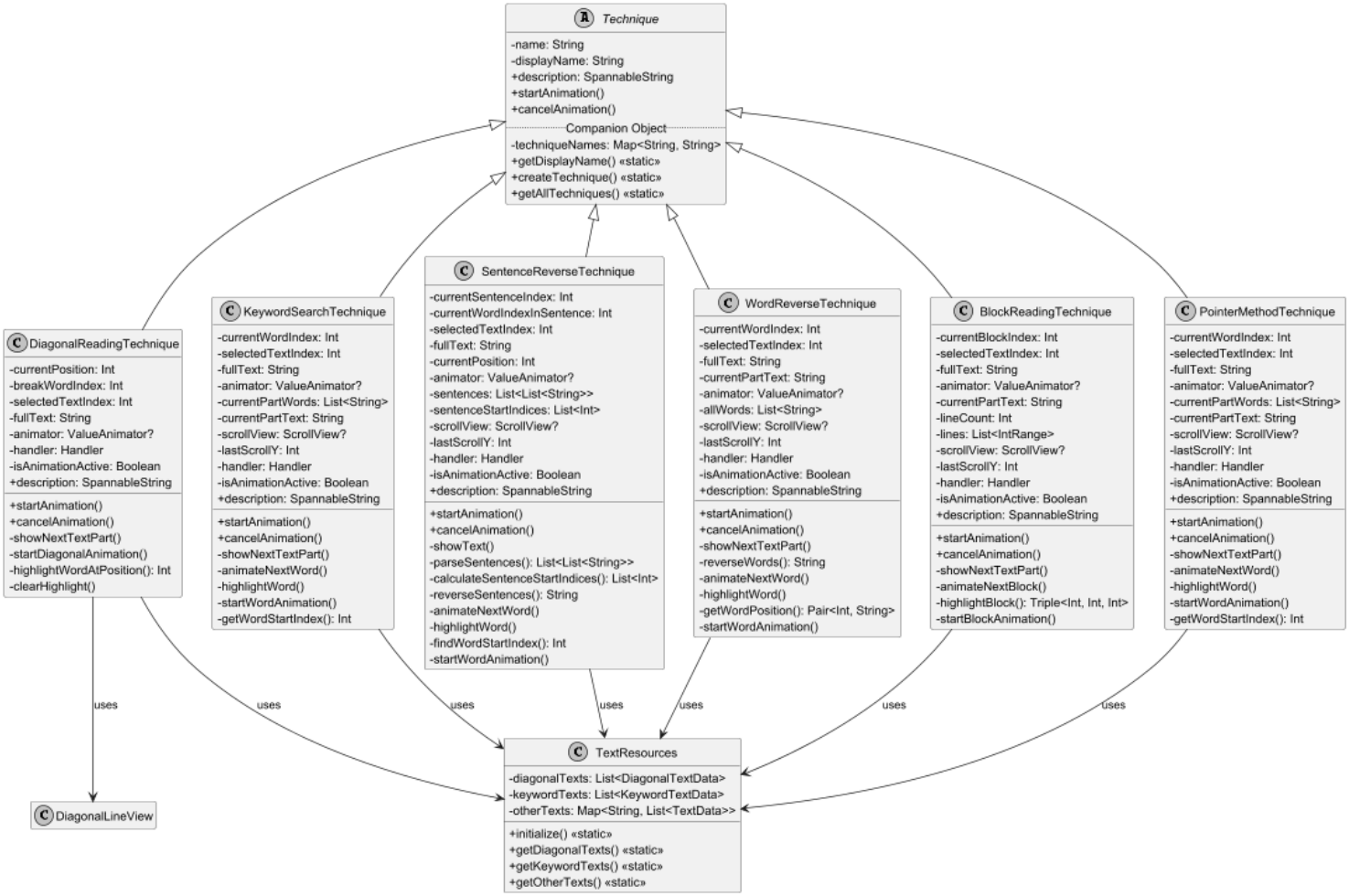

Рис. 10 – Диаграмма классов техник скорочтения и текстовых ресурсов

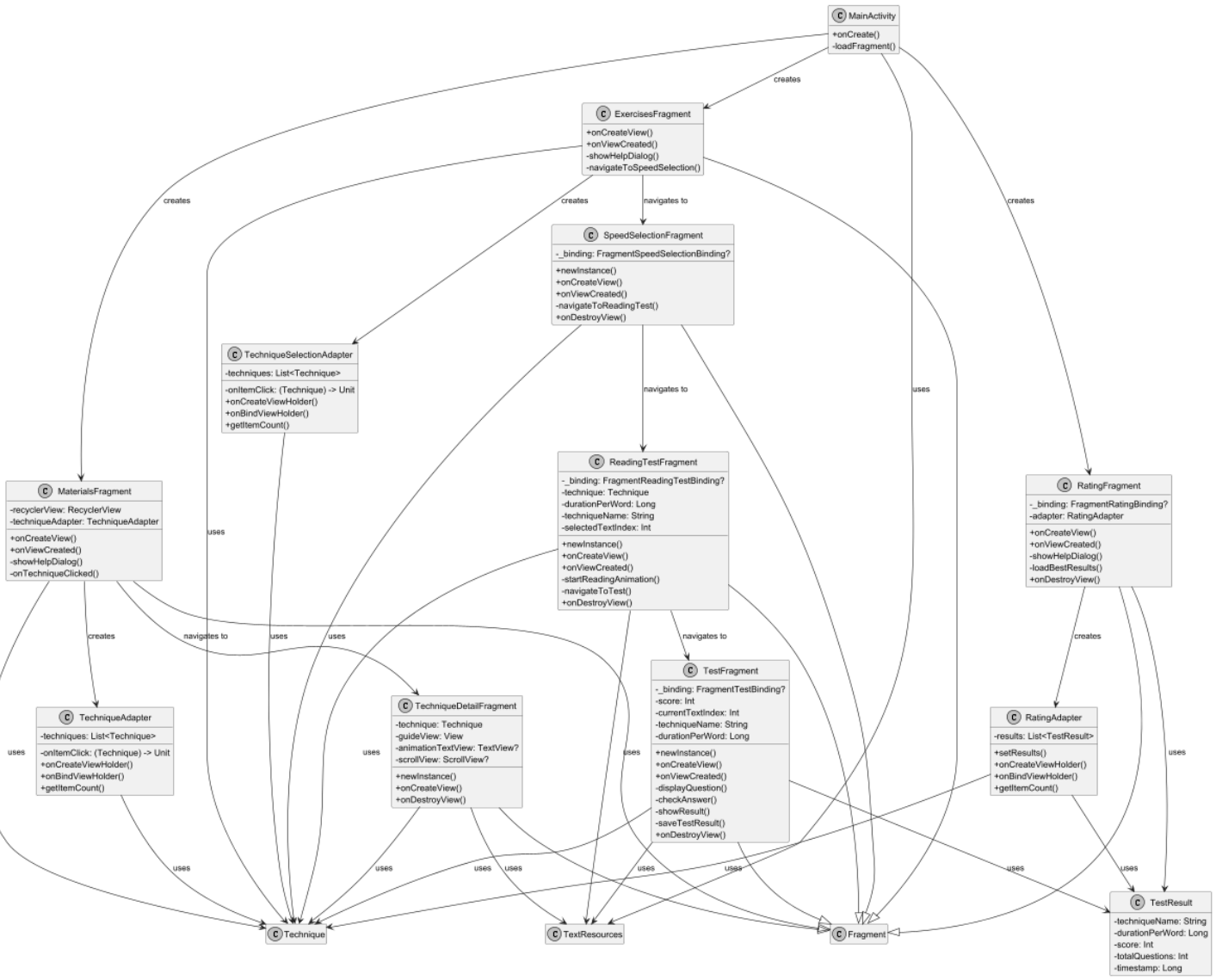

Рис. 11 – Диаграмма классов пользовательского интерфейса и логики приложения

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Приложение Scorochenie разработано для платформы Android с использованием языка программирования Kotlin и среды разработки Android Studio. Приложение реализует функциональность тренировки скорочтения с использованием различных техник, поддерживает анимации текста, настройку скорости и отслеживание прогресса пользователя.

3.1 Архитектура приложения

Главные элементы архитектуры:

1. **MainActivity** — управляющий компонент, координирующий навигацию между фрагментами;
2. **Fragment-структура**:
   — ExercisesFragment — выбор техники тренировки;
   — MaterialsFragment — просмотр материалов;
   — SpeedSelectionFragment — выбор скорости;
   — ReadingTestFragment — отображение анимации текста;
   — TechniqueDetailFragment — описание техники;
3. **BottomNavigationView** — нижняя панель навигации;
4. **TextResources** — централизованное хранилище текстов и вспомогательных данных.

Навигация осуществляется через FragmentManager с добавлением в стек (addToBackStack), что позволяет пользователю возвращаться к предыдущим экранам.

3.2 Основные классы и их функциональность

1. MainActivity

Класс MainActivity является центральной точкой входа приложения. Отвечает за инициализацию пользовательского интерфейса и организацию навигации.

— onCreate(Bundle?) — основной метод жизненного цикла активности. Настраивает макет activity_main.xml, находит BottomNavigationView и задаёт обработчики выбора пунктов меню.

— loadFragment(Fragment) — выполняет замену текущего фрагмента в контейнере fragment_container, упрощая навигацию.

— Инициализирует TextResources, обеспечивая доступ к текстовым материалам для всех техник.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val bottomNavigation =
findViewById<BottomNavigationView>(R.id.bottom_navigation)
        bottomNavigation.setOnItemSelectedListener { item ->
            when (item.itemId) {
                R.id.nav_rating -> {
                    loadFragment(RatingFragment())
                    true
                }
                R.id.nav_exercises -> {
                    loadFragment(ExercisesFragment())
                    true
                }
                R.id.nav_materials -> {
                    loadFragment(MaterialsFragment())
                    true
                }
                else -> false
            }
        }

        if (savedInstanceState == null) {
            loadFragment(RatingFragment())
        }
        TextResources.initialize(this)
    }

    private fun loadFragment(fragment: Fragment) {
        supportFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, fragment)
            .commit()
    }
}
```

2. ExercisesFragment

ExercisesFragment предоставляет интерфейс для выбора техники скорочтения.

— onCreateView(...) — создаёт макет интерфейса на основе fragment_exercises.xml, настраивает RecyclerView, добавляет вертикальные разделители через DividerItemDecoration, подключает адаптер TechniqueSelectionAdapter.

— onViewCreated(...) — обрабатывает клик на иконку справки, выводя поясняющий AlertDialog.

— navigateToSpeedSelection(techniqueName: String) — инициирует переход к SpeedSelectionFragment, передаёт имя выбранной техники.

```
class ExercisesFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_exercises, container,
false)

        val recyclerView = view.findViewById<RecyclerView>(R.id.exercises_list)
        recyclerView.layoutManager = LinearLayoutManager(context)
        val dividerItemDecoration = DividerItemDecoration(
            recyclerView.context,
            LinearLayoutManager.VERTICAL
        )
        recyclerView.addItemDecoration(dividerItemDecoration)

        val techniques = Technique.getAllTechniques()

        recyclerView.adapter = TechniqueSelectionAdapter(techniques) { technique
->
            navigateToSpeedSelection(technique.name)
        }

        return view
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val helpIcon = view.findViewById<ImageView>(R.id.exercises_help_icon)
        helpIcon.setOnClickListener {
            showHelpDialog("Здесь вы найдете список упражнений для тренировки
скорочтения. Погрузитесь в процесс и развивайте свои навыки чтения:\n" +
                    "1. Выберите технику: Определите, какую технику скорочтения
вы хотите практиковать.\n" +
                    "2. Настройте скорость: Установите комфортный темп чтения
для эффективной тренировки.\n" +
                    "3. Прочитайте текст: Внимательно ознакомьтесь с
предложенным текстом.\n" +
                    "4. Ответьте на вопросы: Проверьте понимание материала,
ответив на вопросы по тексту.")
        }
    }

    private fun showHelpDialog(message: String) {
```

```
        android.app.AlertDialog.Builder(requireContext())
            .setTitle("Справка")
            .setMessage(message)
            .setPositiveButton("OK", null)
            .show()
    }

    private fun navigateToSpeedSelection(techniqueName: String) {
        val fragment = SpeedSelectionFragment.newInstance(techniqueName)
        parentFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, fragment)
            .addToBackStack(null)
            .commit()
    }
}
```

3. MaterialsFragment

Фрагмент MaterialsFragment отображает список доступных техник с их описанием.

— onCreateView(...) — настраивает RecyclerView с TechniqueAdapter, заполняет его списком из Technique.getAllTechniques().

— onViewCreated(...) — реализует всплывающую справку при нажатии на иконку.

— onTechniqueClicked(Technique) — осуществляет переход к TechniqueDetailFragment, передавая имя выбранной техники.

```
class MaterialsFragment : Fragment() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var techniqueAdapter: TechniqueAdapter

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_materials, container,
false)

        recyclerView = view.findViewById(R.id.materials_list)
        recyclerView.layoutManager = LinearLayoutManager(context)

        val dividerItemDecoration = DividerItemDecoration(
            recyclerView.context,
            LinearLayoutManager.VERTICAL
        )
        recyclerView.addItemDecoration(dividerItemDecoration)

        val techniques = Technique.getAllTechniques()

        techniqueAdapter = TechniqueAdapter(techniques) { technique ->
            onTechniqueClicked(technique)
```

```
        }
        recyclerView.adapter = techniqueAdapter

        return view
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val helpIcon = view.findViewById<ImageView>(R.id.materials_help_icon)
        helpIcon.setOnClickListener {
            showHelpDialog(
                "Этот раздел создан, чтобы вы могли глубже изучить техники
скорочтения и выбрать те, которые вам подходят! Ознакомьтесь с материалами,
чтобы освоить новые навыки:\n" +
                        "1. Просмотрите, как работают техники: Узнайте, как
каждая техника помогает читать быстрее и лучше понимать текст.\n" +
                        "2. Прочитайте описания техник: Изучите подробные
инструкции и примеры, чтобы применять техники на практике.\n" +
                        "3. Начните изучение: Погрузитесь в материалы и
тренируйтесь, чтобы сделать чтение более эффективным и увлекательным!"
            )
        }
    }

    private fun showHelpDialog(message: String) {
        context?.let {
            AlertDialog.Builder(it)
                .setTitle("Справка")
                .setMessage(message)
                .setPositiveButton("OK", null)
                .show()
        }
    }

    private fun onTechniqueClicked(technique: Technique) {
        val detailFragment = TechniqueDetailFragment.newInstance(technique.name)
        parentFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, detailFragment)
            .addToBackStack(null)
            .commit()
    }
}
```

4. SpeedSelectionFragment

Предоставляет пользователю возможность выбрать скорость отображения текста перед началом тренировки.

— newInstance(...) — статический метод, создающий экземпляр фрагмента с аргументом — именем техники.

— onCreateView(...) — использует View Binding для безопасного доступа к элементам макета fragment_speed_selection.xml.

— onViewCreated(...) — отображает название выбранной техники, устанавливает обработчики кнопок скорости (медленная — 200 мс/слово, средняя — 400 мс, быстрая — 600 мс), вызывает navigateToReadingTest(...).

— navigateToReadingTest(...) — запускает ReadingTestFragment, передаёт параметры.

```
class SpeedSelectionFragment : Fragment() {

    companion object {
        private const val ARG_TECHNIQUE_NAME = "technique_name"

        fun newInstance(techniqueName: String): SpeedSelectionFragment {
            return SpeedSelectionFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_TECHNIQUE_NAME, techniqueName)
                }
            }
        }
    }

    private var _binding: FragmentSpeedSelectionBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentSpeedSelectionBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val techniqueName = arguments?.getString(ARG_TECHNIQUE_NAME) ?: ""
        val techniqueDisplayName = Technique.getDisplayName(techniqueName)
        binding.tvTechniqueTitle.text = techniqueDisplayName

        binding.btnSlowSpeed.setOnClickListener {
            navigateToReadingTest(techniqueName, 200L)
        }
        binding.btnMediumSpeed.setOnClickListener {
            navigateToReadingTest(techniqueName, 400L)
        }
        binding.btnFastSpeed.setOnClickListener {
            navigateToReadingTest(techniqueName, 600L)
        }
    }

    private fun navigateToReadingTest(techniqueName: String, durationPerWord:
Long) {
        val fragment = ReadingTestFragment.newInstance(techniqueName,
durationPerWord)
        parentFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, fragment)
            .addToBackStack(null)
```

```
            .commit()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

5. TechniqueAdapter и TechniqueSelectionAdapter

Адаптеры для RecyclerView, обеспечивающие отображение списка техник в MaterialsFragment и ExercisesFragment соответственно.

— onCreateViewHolder(...) — создаёт элементы списка.

— onBindViewHolder(...) — отображает имя техники и задаёт обработчик клика.

— Используют макет android.R.layout.simple_list_item_1, отображают названия техник через technique.displayName.

```
class TechniqueAdapter(
    private val techniques: List<Technique>,
    private val onItemClick: (Technique) -> Unit // Callback для обработки
кликов
) : RecyclerView.Adapter<TechniqueAdapter.TechniqueViewHolder>() {

    class TechniqueViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        val techniqueName: TextView = itemView.findViewById(android.R.id.text1)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
TechniqueViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(android.R.layout.simple_list_item_1, parent, false)
        return TechniqueViewHolder(view)
    }

    override fun onBindViewHolder(holder: TechniqueViewHolder, position: Int) {
        val technique = techniques[position]
        holder.techniqueName.text = technique.displayName
        holder.itemView.setOnClickListener {
            onItemClick(technique)
        }
    }

    override fun getItemCount(): Int = techniques.size
}

class TechniqueSelectionAdapter(
    private val techniques: List<Technique>,
    private val onItemClick: (Technique) -> Unit
) : RecyclerView.Adapter<TechniqueSelectionAdapter.TechniqueViewHolder>() {
```

```
    class TechniqueViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        val techniqueName: TextView = itemView.findViewById(android.R.id.text1)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
TechniqueViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(android.R.layout.simple_list_item_1, parent, false)
        return TechniqueViewHolder(view)
    }

    override fun onBindViewHolder(holder: TechniqueViewHolder, position: Int) {
        val technique = techniques[position]
        holder.techniqueName.text = technique.displayName
        holder.itemView.setOnClickListener {
            onItemClick(technique)
        }
    }

    override fun getItemCount(): Int = techniques.size
}
```

6. Technique

Абстрактный базовый класс, описывающий общую структуру любой техники скорочтения.

— val name: String, val displayName: String — идентификаторы и наименования для интерфейса.

— val description: SpannableString — форматированное описание, отображаемое в TechniqueDetailFragment.

— fun startAnimation(...) — абстрактный метод, реализующий отображение текста и анимацию для конкретной техники.

— fun cancelAnimation() — метод для корректной остановки анимации.

— companion object:

- getAllTechniques() — возвращает список всех техник;
- createTechnique(name: String) — создаёт экземпляр соответствующей техники;
- getDisplayName(name: String) — получает пользовательское название по идентификатору.

```
abstract class Technique(val name: String, val displayName: String) {
    abstract val description: SpannableString
    abstract fun startAnimation(
        textView: TextView,
        guideView: View,
```

```
        durationPerWord: Long,
        selectedTextIndex: Int,
        onAnimationEnd: () -> Unit
    )

    open fun cancelAnimation() {}
    companion object {
        private val techniqueNames = mapOf(
            "BlockReadingTechnique" to "Чтение \"блоками\"",
            "DiagonalReadingTechnique" to "Чтение по диагонали",
            "KeywordSearchTechnique" to "Поиск ключевых слов",
            "PointerMethodTechnique" to "Метод \"указки\"",
            "SentenceReverseTechnique" to "Предложения наоборот",
            "WordReverseTechnique" to "Слова наоборот"
        )

        fun getDisplayName(name: String): String {
            return techniqueNames[name] ?: name
        }
        fun createTechnique(name: String): Technique {
            val displayName = getDisplayName(name)
            return when (name) {
                "BlockReadingTechnique" -> BlockReadingTechnique()
                "DiagonalReadingTechnique" -> DiagonalReadingTechnique()
                "KeywordSearchTechnique" -> KeywordSearchTechnique()
                "PointerMethodTechnique" -> PointerMethodTechnique()
                "SentenceReverseTechnique" -> SentenceReverseTechnique()
                "WordReverseTechnique" -> WordReverseTechnique()
                else -> object : Technique("UnknownTechnique", displayName) {
                    override val description: SpannableString
                        get() = SpannableString("Описание для этой техники
недоступно")

                    override fun startAnimation(
                        textView: TextView,
                        guideView: View,
                        durationPerWord: Long,
                        selectedTextIndex: Int,
                        onAnimationEnd: () -> Unit
                    ) {
                        textView.text = "Анимация недоступна"
                        guideView.visibility = View.INVISIBLE
                        onAnimationEnd()
                    }

                    override fun cancelAnimation() {
                    }
                }
            }
        }

        fun getAllTechniques(): List<Technique> {
            return techniqueNames.keys.map { createTechnique(it) }
        }
    }
}
```

7. DiagonalReadingTechnique

Реализует технику «Чтение по диагонали». Суть метода заключается в последовательном отображении фрагментов текста с движущейся по диагонали красной линией.

— startAnimation(...) — инициализирует текст, включает движение направляющей линии и подсветку слов.

— showNextTextPart(...) — отображает следующую часть текста, используя ключевые слова как разделители.

— startDiagonalAnimation(...) — анимирует движение линии по диагонали через ValueAnimator.

— highlightWordAtPosition(...) — определяет ближайшее слово к текущей позиции линии и подсвечивает его.

— cancelAnimation() — прерывает анимацию, очищает ресурсы.

```
class DiagonalReadingTechnique : Technique("DiagonalReadingTechnique", "Чтение
по диагонали") {
    private var currentPosition = 0
    private var breakWordIndex = 0
    private var selectedTextIndex = 0
    private var fullText: String = ""
    private var animator: ValueAnimator? = null
    private val handler = Handler(Looper.getMainLooper())
    private var isAnimationActive = false

    override val description: SpannableString
        get() {
            val text = "Чтение по диагонали — это способ быстрого ознакомления с
текстом, при котором взгляд скользит сверху вниз по диагонали, захватывая общую
структуру и главные элементы.\n" +
                    "Вместо того чтобы читать каждое слово, вы охватываете
страницу бегло, выхватывая смысловые опоры — такие как начальные и конечные
слова абзацев, цифры или повторы.\n" +
                    "Этот метод позволяет быстро получить общее представление о
содержании и решить, стоит ли читать подробнее."
            val spannable = SpannableString(text)
            spannable.setSpan(StyleSpan(android.graphics.Typeface.BOLD), 0,
name.length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            spannable.setSpan(StyleSpan(android.graphics.Typeface.BOLD),
text.indexOf("сверху вниз по диагонали"), text.indexOf("сверху вниз по
диагонали") + "сверху вниз по диагонали".length,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            spannable.setSpan(StyleSpan(android.graphics.Typeface.BOLD),
text.indexOf("смысловые опоры"), text.indexOf("смысловые опоры") + "смысловые
опоры".length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            spannable.setSpan(StyleSpan(android.graphics.Typeface.BOLD),
text.indexOf("начальные и конечные"), text.indexOf("начальные и конечные") +
"начальные и конечные".length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            return spannable
        }

    override fun startAnimation(
        textView: TextView,
```

```
        guideView: View,
        durationPerWord: Long,
        selectedTextIndex: Int,
        onAnimationEnd: () -> Unit
    ) {
        this.selectedTextIndex = selectedTextIndex
        fullText =
TextResources.getDiagonalTexts().getOrNull(selectedTextIndex)?.text?.replace("\n
", " ") ?: ""
        currentPosition = 0
        breakWordIndex = 0
        isAnimationActive = true

        val safeDurationPerWord = if (durationPerWord <= 0) 400L else
durationPerWord
        val wordDurationMs = (60_000 / safeDurationPerWord).coerceAtLeast(50L)

        guideView.visibility = View.INVISIBLE

        textView.gravity = android.view.Gravity.TOP
        textView.isSingleLine = false
        textView.maxLines = Int.MAX_VALUE

        handler.post {
            if (isAnimationActive) {
                showNextTextPart(textView, guideView, wordDurationMs,
onAnimationEnd)
            }
        }
    }

    private fun showNextTextPart(
        textView: TextView,
        guideView: View,
        wordDurationMs: Long,
        onAnimationEnd: () -> Unit
    ) {
        if (!isAnimationActive) return

        if (currentPosition >= fullText.length) {
            guideView.visibility = View.INVISIBLE
            animator?.cancel()
            clearHighlight(textView)
            if (isAnimationActive) onAnimationEnd()
            return
        }

        val currentBreakWords =
TextResources.getDiagonalTexts().getOrNull(selectedTextIndex)?.breakWords ?:
emptyList()
        val breakWord = if (breakWordIndex < currentBreakWords.size)
currentBreakWords[breakWordIndex] else ""
        val breakPosition = if (breakWord.isNotEmpty()) {
            val index = fullText.indexOf(breakWord, currentPosition)
            if (index == -1) fullText.length else index + breakWord.length
        } else {
            fullText.length
        }

        val partText = fullText.substring(currentPosition, breakPosition).trim()

        textView.text = partText
        textView.visibility = View.VISIBLE
```

```
        handler.post {
            if (!isAnimationActive) return@post
            val parent = textView.parent as View
            val diagonalLineView =
parent.findViewById<DiagonalLineView>(R.id.diagonal_line_view)
            if (diagonalLineView != null) {
                diagonalLineView.requestLayout()
                startDiagonalAnimation(textView, guideView, breakPosition,
partText, wordDurationMs, onAnimationEnd)
            } else {
                if (isAnimationActive) onAnimationEnd()
            }
        }
    }

    private fun startDiagonalAnimation(
        textView: TextView,
        guideView: View,
        newPosition: Int,
        partText: String,
        wordDurationMs: Long,
        onAnimationEnd: () -> Unit
    ) {
        if (!isAnimationActive) return

        animator?.cancel()

        val wordCount = partText.split("\\s+".toRegex()).filter {
it.isNotEmpty() }.size
        val totalDuration = wordCount * wordDurationMs

        val layout = textView.layout
        if (layout == null) {
            handler.postDelayed({
                if (isAnimationActive) startDiagonalAnimation(textView,
guideView, newPosition, partText, wordDurationMs, onAnimationEnd)
            }, 50)
            return
        }

        val width = textView.width.toFloat()
        val visibleHeight = textView.height.toFloat()
        val totalLines = layout.lineCount
        val lastLineTop = if (totalLines > 1) layout.getLineTop(totalLines - 1)
else visibleHeight
        val heightExcludingLastLine = if (totalLines > 1) lastLineTop.toFloat()
else visibleHeight

        guideView.visibility = View.INVISIBLE
        guideView.translationX = 0f
        guideView.translationY = 0f

        val initialLine = highlightWordAtPosition(textView, 0f, 0f, -1)

        animator = ValueAnimator.ofFloat(0f, 1f).apply {
            duration = totalDuration
            interpolator = LinearInterpolator()
            var lastLine = initialLine

            addUpdateListener { animation ->
                if (!isAnimationActive) return@addUpdateListener
                val fraction = animation.animatedValue as Float
```

```
                val y = fraction * heightExcludingLastLine
                val x = fraction * width

                guideView.translationX = x - (guideView.width / 2)
                guideView.translationY = y

                val currentLine = highlightWordAtPosition(textView, x, y,
lastLine)
                if (currentLine != -1) lastLine = currentLine
            }
            addListener(
                onEnd = {
                    if (!isAnimationActive) return@addListener
                    clearHighlight(textView)
                    guideView.visibility = View.INVISIBLE
                    currentPosition = newPosition
                    breakWordIndex++
                    showNextTextPart(textView, guideView, wordDurationMs,
onAnimationEnd)
                }
            )
            start()
        }
    }

    private fun highlightWordAtPosition(textView: TextView, x: Float, y: Float,
lastLine: Int): Int {
        if (!isAnimationActive) return -1

        val layout = textView.layout ?: return -1
        val visibleHeight = textView.height.toFloat()

        val adjustedY = y.coerceIn(0f, visibleHeight)
        val currentLine = layout.getLineForVertical(adjustedY.toInt())

        val totalLines = layout.lineCount
        if (currentLine == totalLines - 1 || currentLine <= lastLine) {
            return currentLine
        }

        val diagonalSlope = visibleHeight / textView.width.toFloat()
        val expectedX = adjustedY / diagonalSlope

        var closestOffset = -1
        var minDistance = Float.MAX_VALUE

        for (offset in layout.getLineStart(currentLine) until
layout.getLineEnd(currentLine)) {
            if (textView.text[offset].isWhitespace()) continue

            val charLeft = layout.getPrimaryHorizontal(offset)
            val charRight = if (offset + 1 < textView.text.length)
layout.getPrimaryHorizontal(offset + 1) else charLeft
            var charX = (charLeft + charRight) / 2

            val distance = abs(charX - expectedX)
            if (distance < minDistance) {
                minDistance = distance
                closestOffset = offset
            }
        }

        if (closestOffset != -1) {
```

```
            val text = textView.text.toString()
            var start = closestOffset
            var end = closestOffset

            while (start > 0 && !text[start - 1].isWhitespace()) start--
            while (end < text.length && !text[end].isWhitespace()) end++

            val spannable = SpannableString(text)
            val existingSpans = spannable.getSpans(0, spannable.length,
BackgroundColorSpan::class.java)
            for (span in existingSpans) {
                spannable.removeSpan(span)
            }
            spannable.setSpan(
                BackgroundColorSpan(Color.YELLOW),
                start,
                end,
                Spannable.SPAN_EXCLUSIVE_EXCLUSIVE
            )
            textView.text = spannable
        }

        return currentLine
    }

    private fun clearHighlight(textView: TextView) {
        if (!isAnimationActive) return

        val text = textView.text.toString()
        val spannable = SpannableString(text)
        val existingSpans = spannable.getSpans(0, spannable.length,
BackgroundColorSpan::class.java)
        for (span in existingSpans) {
            spannable.removeSpan(span)
        }
        textView.text = spannable
    }

    override fun cancelAnimation() {
        isAnimationActive = false
        animator?.cancel()
        handler.removeCallbacksAndMessages(null)
    }
}
```

8. DiagonalLineView

Кастомный View, используемый для визуализации красной диагональной линии при технике DiagonalReadingTechnique.

— onDraw(...) — рисует линию от верхнего левого до нижнего правого угла.

— onMeasure(...) — синхронизирует высоту View с высотой связанного TextView

```
public class DiagonalLineView @JvmOverloads constructor(
    context: Context,
```

```
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0
) : View(context, attrs, defStyleAttr) {

    private val paint = Paint().apply {
        color = Color.RED
        strokeWidth = 4f * resources.displayMetrics.density // 4dp
        style = Paint.Style.STROKE
        isAntiAlias = true
    }

    override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec)
        val width = measuredWidth
        val textView = (parent as
View).findViewById<TextView>(R.id.animation_text_diagonal)
        val height = textView?.measuredHeight ?: 0
        setMeasuredDimension(width, height)
        Log.d("DiagonalLineView", "Measured size: ${width}x${height}")

        if (textView != null) {
            textView.addOnLayoutChangeListener { _, _, _, _, _, _, _, _, _ ->
                if (textView.measuredHeight != measuredHeight) {
                    requestLayout()
                }
            }
        }
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        canvas.drawLine(0f, 0f, width.toFloat(), height.toFloat(), paint)
        Log.d("DiagonalLineView", "Drawing line with size: ${width}x${height}")
    }
}
```

9. PointerMethodTechnique

Реализует технику «Метод указки». Текст подсвечивается по одному слову, имитируя ведение строки указкой.

— startAnimation(...) — подготавливает текст, запускает поочерёдную подсветку слов.

— showNextTextPart(...) — формирует список слов, готовых для показа.

— highlightWord(...) — выделяет текущее слово цветом.

— startWordAnimation(...) — анимирует перемещение направляющего объекта вдоль строки, обеспечивает автопрокрутку через ScrollView.

— cancelAnimation() — останавливает анимацию и удаляет все отложенные действия.

```
class PointerMethodTechnique : Technique("PointerMethodTechnique", "Метод
указки") {
    private var currentWordIndex = 0
```

```
    private var selectedTextIndex = 0
    private var fullText: String = ""
    private var animator: ValueAnimator? = null
    private var currentPartWords: List<String> = emptyList()
    private var currentPartText: String = ""
    private var scrollView: ScrollView? = null
    private var lastScrollY: Int = 0
    private val handler = Handler(Looper.getMainLooper())
    private var isAnimationActive = false

    override val description: SpannableString
        get() {
            val text = "Метод \"указки\" — это техника скорочтения, в которой используется визуальное сопровождение текста для направления внимания. Вместо пальца или ручки, в приложении слова подсвечиваются по очереди, помогая глазам двигаться по строкам без остановок и возвратов.\n" +
                    "Такая подача помогает удерживать ритм чтения и повышает концентрацию на ключевых фразах.\n" +
                    "Следите за подсвеченными словами и старайтесь воспринимать информацию с их скоростью — это способствует более быстрому и осознанному чтению."
            val spannable = SpannableString(text)
            spannable.setSpan(StyleSpan(Typeface.BOLD), 0, name.length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            spannable.setSpan(StyleSpan(Typeface.BOLD), text.indexOf("визуальное сопровождение текста"), text.indexOf("визуальное сопровождение текста") + "визуальное сопровождение текста".length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            spannable.setSpan(StyleSpan(Typeface.BOLD), text.indexOf("за подсвеченными словами"), text.indexOf("за подсвеченными словами") + "за подсвеченными словами".length, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
            return spannable
        }

    override fun startAnimation(
        textView: TextView,
        guideView: View,
        durationPerWord: Long,
        selectedTextIndex: Int,
        onAnimationEnd: () -> Unit
    ) {
        this.selectedTextIndex = selectedTextIndex
        fullText = TextResources.getOtherTexts()["Метод указки"]?.getOrNull(selectedTextIndex)?.text?.replace("\n", " ") ?: ""
        currentWordIndex = 0
        lastScrollY = 0
        isAnimationActive = true

        val safeDurationPerWord = if (durationPerWord <= 0) 400L else durationPerWord
        val wordDurationMs = (60_000 / safeDurationPerWord).coerceAtLeast(50L)

        scrollView = textView.parent as? ScrollView

        textView.gravity = android.view.Gravity.TOP
        textView.isSingleLine = false
        textView.maxLines = Int.MAX_VALUE
        handler.post {
            if (isAnimationActive) {
                showNextTextPart(textView, guideView, wordDurationMs, onAnimationEnd)
            }
        }
    }
```

```
    private fun showNextTextPart(
        textView: TextView,
        guideView: View,
        wordDurationMs: Long,
        onAnimationEnd: () -> Unit
    ) {
        if (!isAnimationActive) return

        currentPartText = fullText
        currentPartWords = currentPartText.split("\\s+".toRegex()).filter {
it.isNotEmpty() }
        currentWordIndex = 0

        textView.text = currentPartText
        animateNextWord(textView, guideView, wordDurationMs, onAnimationEnd)
    }

    private fun animateNextWord(
        textView: TextView,
        guideView: View,
        wordDurationMs: Long,
        onAnimationEnd: () -> Unit
    ) {
        if (!isAnimationActive) return

        if (currentWordIndex >= currentPartWords.size) {
            guideView.visibility = View.INVISIBLE
            animator?.cancel()
            textView.text = currentPartText
            if (isAnimationActive) onAnimationEnd()
            return
        }

        highlightWord(textView)
        startWordAnimation(textView, guideView, wordDurationMs, onAnimationEnd)
    }

    private fun highlightWord(textView: TextView) {
        if (!isAnimationActive) return

        val spannable = SpannableString(currentPartText)
        val existingSpans = spannable.getSpans(0, spannable.length,
BackgroundColorSpan::class.java)
        for (span in existingSpans) {
            spannable.removeSpan(span)
        }
        var startIndex = 0
        var wordCount = 0

        currentPartWords.forEach { word ->
            if (wordCount == currentWordIndex) {
                val endIndex = startIndex + word.length
                spannable.setSpan(
                    BackgroundColorSpan(Color.YELLOW),
                    startIndex,
                    endIndex,
                    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE
                )
            }
            startIndex += word.length
            if (startIndex < currentPartText.length &&
currentPartText[startIndex] == ' ') {
```

```
                startIndex++
            }
            wordCount++
        }

        textView.text = spannable
    }
    private fun startWordAnimation(
        textView: TextView,
        guideView: View,
        wordDurationMs: Long,
        onAnimationEnd: () -> Unit
    ) {
        if (!isAnimationActive) return
        guideView.visibility = View.INVISIBLE
        animator?.cancel()

        val layout = textView.layout
        if (layout == null) {
            handler.postDelayed({
                if (isAnimationActive) animateNextWord(textView, guideView,
wordDurationMs, onAnimationEnd)
            }, 200)
            return
        }
        val wordStartIndex = getWordStartIndex(currentWordIndex,
currentPartText)
        val wordEndIndex = wordStartIndex +
currentPartWords[currentWordIndex].length

        if (wordStartIndex < 0 || wordStartIndex >= currentPartText.length) {
            currentWordIndex++
            animateNextWord(textView, guideView, wordDurationMs, onAnimationEnd)
            return
        }
        val startLine = layout.getLineForOffset(wordStartIndex)
        val endLine = layout.getLineForOffset(wordEndIndex)
        val startX = layout.getPrimaryHorizontal(wordStartIndex)
        var endX = layout.getPrimaryHorizontal(wordEndIndex)
        if (endX == startX) {
            endX = startX + layout.getPrimaryHorizontal(wordStartIndex + 1)
        }
        val lineTop = layout.getLineTop(startLine).toFloat()
        val lineBottom = layout.getLineBottom(startLine).toFloat()
        val lineY = (lineTop + lineBottom) / 2

        scrollView?.let { sv ->
            handler.post {
                if (!isAnimationActive) return@post
                val scrollViewHeight = sv.height
                val currentScrollY = sv.scrollY
                val lineTopPosition = layout.getLineTop(startLine)
                val lineBottomPosition = layout.getLineBottom(startLine)

                val visibleTop = currentScrollY
                val visibleBottom = currentScrollY + scrollViewHeight * 2 / 3

                if (lineTopPosition < visibleTop || lineBottomPosition >
visibleBottom) {
                    val targetScrollY = (lineTopPosition - scrollViewHeight /
3).coerceAtLeast(0).toInt()
                    if (targetScrollY != lastScrollY) {
                        ValueAnimator.ofInt(currentScrollY, targetScrollY).apply
```

```
{
                            duration = wordDurationMs / 2
                            addUpdateListener { animation ->
                                val value = animation.animatedValue as Int
                                sv.scrollTo(0, value)
                            }
                            addListener(
                                onEnd = {
                                    lastScrollY = targetScrollY
                                }
                            )
                            start()
                        }
                    }
                }
            }
        }
        animator = ValueAnimator.ofFloat(0f, 1f).apply {
            duration = wordDurationMs
            addUpdateListener { animation ->
                if (!isAnimationActive) return@addUpdateListener
                val fraction = animation.animatedValue as Float
                val currentX = startX + (endX - startX) * fraction
                guideView.translationX = currentX - (guideView.width / 2) +
textView.left
                guideView.translationY = lineY + textView.top.toFloat() -
(scrollView?.scrollY?.toFloat() ?: 0f)
            }
            addListener(
                onEnd = {
                    if (isAnimationActive) {
                        currentWordIndex++
                        animateNextWord(textView, guideView, wordDurationMs,
onAnimationEnd)
                    }
                }
            )
            start()
        }
    }
    private fun getWordStartIndex(wordIndex: Int, text: String): Int {
        var startIndex = 0
        var count = 0
        text.split("\\s+".toRegex()).forEachIndexed { index, word ->
            if (count == wordIndex) {
                return startIndex
            }
            startIndex += word.length
            if (startIndex < text.length && text[startIndex] == ' ') {
                startIndex++
            }
            count++
        }
        return startIndex
    }
    override fun cancelAnimation() {
        isAnimationActive = false
        animator?.cancel()
        handler.removeCallbacksAndMessages(null)
    }
}
```

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы разработано мобильное приложение для операционной системы Android, предназначенное для тренировки навыков скорочтения. Работа направлена на решение актуальной задачи - повышение скорости и качества восприятия текстовой информации в условиях роста информационных потоков. Достижение поставленной цели обеспечено последовательным выполнением всех сформулированных задач.

В рамках изучения предметной области проанализированы особенности скорочтения и ключевые техники, такие как чтение блоками, чтение по диагонали, поиск ключевых слов, метод указки, чтение слов и предложений в обратном порядке. Проведён обзор существующих программных продуктов (Quickify, Readmical, Spritz, Spreeder), что позволило выявить их преимущества и недостатки, а также определить функциональные требования к разрабатываемому приложению.

На этапе проектирования создана функциональная модель, включающая сценарии выбора техник, настройки параметров, выполнения упражнений и просмотра статистики. Разработаны эскизы экранов пользовательского интерфейса с использованием XML-макетов в среде Android Studio, обеспечивающие интуитивную навигацию и поддержку всех предусмотренных техник. Логическая модель представлена в виде диаграмм классов, описывающих структуру техник скорочтения, текстовых ресурсов и компонентов интерфейса.

Тестирование приложения подтвердило его соответствие поставленным требованиям: корректное отображение техник, точность подсчёта результатов и удобство взаимодействия с интерфейсом. Приложение предоставляет пользователям эффективный инструмент для развития навыков скорочтения, позволяя настраивать параметры тренировок и отслеживать прогресс.

В дальнейшем для развития приложения планируется реализация следующих направлений:

1. Введение системы последовательного открытия техник, где изначально доступна только самая лёгкая техника, а последующие становятся доступны после успешного прохождения предыдущей, что обеспечит постепенное усложнение тренировок и повысит мотивацию пользователей.

2. Расширение набора техник скорочтения, включая:

— Технику «зашумлённый» текст, предполагающую добавление визуальных помех (например, наложение случайных символов или искажений), для тренировки концентрации и выделения значимой информации.

— Технику «текст за шторкой», где текст отображается постепенно через движущуюся область видимости, развивая навык быстрого восприятия ограниченного фрагмента текста.

— Технику «текст с закрытой частью строк», при которой часть строк скрыта, заставляя пользователя предугадывать содержание и улучшать контекстуальное понимание.

3. Добавление настроек для отображения текста, позволяющих пользователю изменять размер текста и цветовую схему для повышения комфорта чтения и адаптации под индивидуальные предпочтения.

4. Реализация гибкой настройки скорости анимации, предоставляющей возможность задавать произвольные значения скорости отображения текста, что обеспечит более точную персонализацию тренировок.

Таким образом, поставленные в работе цели и задачи были достигнуты, а разработанное приложение может служить основой для дальнейшего совершенствования полноценного мобильного решения в области тренировки скорочтения.