Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Институт компьютерных и инженерных наук
Кафедра информационных и управляющих систем
Направление подготовки 09.03.01 – Информатика и вычислительная техника
Направленность (профиль) образовательной программы Информатика и вычислительная техника

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«____» _____ 2025 г.

БАКАЛАВРСКАЯ РАБОТА

на тему: Разработка приложения для создания конфиденциальных заметок

Исполнитель
студент группы 1103-об _____ Д. Б. Ефремов
(подпись, дата)

Руководитель
доцент _____ И. М. Акилова
(подпись, дата)

Консультант
по безопасности и экологичности _____ А.Б. Булгаков
(подпись, дата)
доцент, канд. техн. наук

Нормоконтроль
Инженер кафедры _____ В.Н. Адаменко
(подпись, дата)

Благовещенск 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Институт компьютерных и инженерных наук
Кафедра информационных и управлявших систем

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Бушманов
«____» _____ 2024 г.

ЗАДАНИЕ

К бакалаврской работе студента Ефремов Данил Борисович

1.Тема выпускной квалификационной работы: Разработка приложения для создания кондифициальных заметок

(утверждено приказом от 14.04.2025 № 980-уч)

2. Срок сдачи студентом законченной работы: 10.06.2025

3. Содержание бакалаврской работы (перечень подлежащих разработке вопросов): анализ объекта исследования; разработка программного продукта; проектирование программного продукта; безопасность и экологичность.

4. Перечень материалов приложения (наличие чертежей, таблиц, графиков, схем, программных продуктов, иллюстративного материала и т.п.):

5. Консультанты по выпускной квалификационной работе: консультант по безопасности и экологичности: доцент, канд. техн. наук А.Б. Булгаков

6. Дата выдачи задания: 10.10.2024г.

Руководитель бакалаврской работы: доцент доцент, И.М. Акилова

Задание принял к исполнению (10.10.2024): _____

(подпись студента)

РЕФЕРАТ

Отчет по практической работе содержит 65 с., 21 рисунков, 5 таблица, 28 источников.

ANDROID, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ШИФРОВАНИЕ, AES, KOTLIN, ANDROID STUDIO.

В рамках бакалаврской работы было разработано мобильное приложение для кондифициальных заметок с использованием языка программирования kotlin. Для реализации функционала безопасности был создан модуль шифрования, основанный на алгоритме AES. В качестве платформы для разработки использовалась среда Android Studio, а для локального хранения данных использовалась СУБД SQLite. Интерфейс приложения реализован с использованием библиотеки jetpack compose, обеспечивающей адаптивный дизайн и высокую производительность.

В результате проделанной работы создано приложение, позволяющее пользователям удобно управлять заметками с поддержкой полной конфиденциальности. Программа включает все основные функции работы с данными: создание, чтение, редактирование и удаление заметок. Все операции сопровождаются автоматическим шифрованием перед сохранением и дешифрованием при открытии. Дополнительно реализован механизм и безопасное управление ключами шифрования.

Разработанное приложение демонстрирует эффективность применения современных методов криптографии в повседневных задачах, обеспечивая высокий уровень защиты данных при минимальном влиянии на удобство использования.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

ПО – Программное обеспечение;

IDE – Интегрированная среда разработки;

GUI – Graphical User Interface (Графический интерфейс пользователя);

UI – пользовательский интерфейс;

БД – база данных

СУБД – система управления базами данных

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном цифровом мире защита персональных данных становится все более актуальной задачей. С ростом количества мобильных устройств и увеличением объема хранимой на них информации возникает необходимость в надежных методах обеспечения конфиденциальности данных пользователей. Одним из уязвимых типов данных являются личные заметки, содержащие важную информацию, такую как пароли, финансовые записи или идеи.

Одним из эффективных способов защиты данных является их шифрование, позволяющее предотвратить несанкционированный доступ к информации. В данной работе рассматривается разработка мобильного приложения для операционной системы Android, которое позволяет пользователям создавать и хранить заметки в зашифрованном виде.

Актуальность данной работы обусловлена растущей необходимостью защиты конфиденциальных данных пользователей мобильных устройств. Большинство стандартных приложений для ведения заметок не обеспечивают достаточного уровня безопасности, а использование сторонних облачных сервисов несет риски утечки информации. Разработка приложения с локальным шифрованием позволит пользователям хранить свои данные в безопасном виде, не зависимо от внешних факторов.

Практической значимость разработанной программы заключается в том, что приложение предоставляет безопасный и удобный инструмент для хранения конфиденциальных данных. Оно будет полезным как для частных пользователей, так и для профессиональной деятельности, где требуется защита информации от утечек и несанкционированного доступа.

Объектом исследования выступает: мобильное приложение для создания, хранения и защиты пользовательских заметок на платформе Android.

Предмет исследования: Методы шифрования данных.

Целью данной работы является разработка мобильного приложения для создания и безопасного хранения текстовых заметок с шифрованием

Для достижения поставленной цели были сформулированы следующие задачи:

- провести анализ методов шифрования;
- выбор инструментов, необходимых в разработке мобильного приложения;
- проектирование архитектуры приложения;
- реализовать механизм шифрования и дешифрования;
- разработать интуитивный интерфейс мобильного приложения.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Шифрование представляет собой метод преобразования информации в закодированный вид, обеспечивающий её конфиденциальность и доступность исключительно для лиц, обладающих ключом. Этот процесс применяется для защиты данных от несанкционированного доступа как при их хранении, так и при передаче через коммуникационные каналы. В зависимости от принципов использования ключей, в криптографии выделяют два основных подхода: симметричное и асимметричное шифрование.

Симметричные методы шифрования используют единый ключ для шифрования и дешифрования, что обеспечивает высокую скорость обработки данных. Это делает их оптимальными для работы с большими объёмами информации, например, для защиты файловых систем или передачи данных в реальном времени. В рамках симметричных методов выделяют блочные и потоковые шифры. Блочные шифры разбивают данные на фиксированные блоки, обрабатывая их через определенное количество раундов преобразований, что повышает криптостойкость. Потоковые шифры шифруют данные побитно, что позволяет минимизировать задержки и подходит для потоковой передачи информации. Однако симметричные системы требуют безопасного канала для передачи ключа, что остаётся их ключевым ограничением.

Асимметричные методы шифрования основаны на паре математически связанных ключей: публичного (для шифрования) и приватного (для дешифрования). Безопасность таких систем основана на сложности решения математических задач, таких как факторизация больших чисел или дискретное логарифмирование. Асимметричные алгоритмы применяются для обмена ключами, цифровых подписей и аутентификации. Несмотря на высокую криптостойкость, они требуют значительных вычислительных ресурсов.

Для разрабатываемого Android-приложения, ориентированного на создание конфиденциальных заметок, оптимальным выбором является симметричный метод шифрования. Это обусловлено следующими преимуществами:

− Высокая скорость и эффективность обработки данных — симметричные алгоритмы обеспечивают мгновенное шифрование и дешифрование даже объёмных текстовых заметок благодаря блочным методам обработки. Это критично для мобильных приложений, где важны отзывчивость интерфейса.

− Минимальная нагрузка на ресурсы — в отличие от асимметричных методов, симметричные требуют меньше вычислительной мощности, что экономит заряд батареи и подходит для устройств с ограниченными ресурсами.

− Безопасность ключа — ключ не сохраняется в памяти устройства, а каждый раз генерируется из введённого пользователем пароля. Это исключает риск его компрометации через утечку из памяти.

В рамках дипломной работы были проанализированы четыре алгоритма: DES, 3DES, AES и Twofish. Их выбор обусловлен широким применением в прошлом и настоящем, а также различиями в структуре, скорости и уровне безопасности.

DES (Data Encryption Standard) был стандартизирован в 1977 году и стал первым массовым симметричным алгоритмом. Он оперирует блоками по 64 бита и использует 56-битный ключ, реализуя 16 раундов шифрования на основе сети Фейстеля. Однако его криптографическая стойкость оказалась недостаточной: в 1999 году DES был взломан за 22 часа с помощью перебора всех возможных ключей (2^{56} комбинаций). Также алгоритм уязвим к дифференциальному и линейному криптоанализу, что привело к его отмене NIST в 2005 году.

3DES (Triple DES) стал попыткой модернизировать DES, применяя тройное шифрование с двумя или тремя ключами. Эффективная длина ключа увеличена до 112 или 168 бит, но размер блока остался 64 бита. При этом скорость шифрования снизилась в 2–3 раза по сравнению с DES (около 5–10 МБ/с), а уязвимость к атаке *meet-in-the-middle* сохранилась.

AES (Advanced Encryption Standard) был принят в 2001 году как замена DES и стал основным стандартом шифрования. Поддерживая блоки по 128 бит и ключи длиной 128, 192 или 256 бит, AES реализует 10–14 раундов в зависимости от длины ключа. Благодаря структуре подстановки, перестановки и использовании операций XOR, он достигает скорости до 100Мб/с. На данный момент AES считается безопасным: не известны фактические случаи взлома, а теоретические атаки требуют сложности, превышающей 2^{128} операций. Алгоритм широко используется в TLS/SSL, дисковом шифровании, Wi-Fi и блокчейн-технологиях.

Twofish, разработанный Брюсом Шнайером, стал финалистом конкурса AES. Он использует блоки по 128 бит и ключи до 256 бит, реализуя 16 раундов с ключезависимыми S-блоками и гибридной структурой. Скорость шифрования составляет 20–50 МБ/с, что ниже, чем у AES, но компенсируется высокой стойкостью к линейному и дифференциальному криптоанализу. На данный момент неизвестны практические атаки на Twofish, однако его сложная реализация и отсутствие стандартизации ограничивают применение в массовых продуктах.

Сравнительный анализ выявил ключевые различия между алгоритмами. DES и 3DES уступают современным требованиям из-за малого размера блока (64 бита) и уязвимостей к перебору. AES демонстрирует оптимальное соотношение скорости и безопасности, что делает его предпочтительным для новых систем.

1.1 Метод шифрования AES

Шифрование Advanced Encryption Standard (AES) представляет собой симметричный алгоритм блочного шифрования, разработанный в рамках конкурса, организованного Национальным институтом стандартов и технологий США (NIST). Алгоритм был официально утвержден в качестве федерального стандарта FIPS 197 в 2001 году. AES был разработан как замена устаревшего алгоритма DES, который перестал соответствовать современным требованиям безопасности в связи с ограниченной длиной ключа, составляющий 56 бит, и

уязвимостью к атакам методом полного перебора. В отличие от DES, AES обеспечивает значительно более высокий уровень криптостойкости за счет поддержки ключей длиной 128, 192 и 256 бит. Алгоритм характеризуется высокой производительностью как при программной, так и при аппаратной реализации, а также универсальностью применения, что делает его одним из наиболее распространенных стандартов защиты данных в современных информационных системах.

Стандарт шифрования AES представлен тремя вариантами алгоритма: AES-128, AES-192 и AES-256. Основные различия между ними определяются длиной ключа и количеством раундов шифрования. Эти характеристики регулируются параметрами:

Nb — параметр, указывающий длину блока данных. Nb всегда равен 4, что соответствует 128-битному блоку данных (16 байт). Данные организуются в матрицу State размером 4×4 байта, где каждая строка — 32-битное слово.

Nk— параметр, задающий длину исходного ключа шифрования в 32-битных словах. Этот параметр напрямую влияет на генерацию раундовых ключей

Nr — количество раундов шифрования

Каждый из вариантов AES использует уникальную комбинацию этих параметров, что обеспечивает разный уровень криптографической стойкости. Комбинации для каждой версии представлено в таблице 1.

Таблица 1 – Значение параметров в соответствии с версиями шифрования

Версия AES	Nb	Nk	Nr
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

1.1.1 Математические основы полей Галуа

AES (Advanced Encryption Standard) использует в своей основе конечные поля, также известные как поля Галуа. Эти математические структуры играют

ключевую роль в обеспечении безопасности алгоритма, поскольку они обеспечивают строгие правила для арифметических операций, гарантируя обратимость и нелинейность.

Поле Галуа — это поле с конечным числом элементов, порядок которого равен степени простого числа p^n, где p — простое число, являющийся характеристика поля, а n — натуральное число, обозначающее максимальное количество элементов. Такие поля обозначаются как $GF(p^n)$. В случае AES используется поле $GF(2^8)$, то есть поле с 2^8 равно 256 элементами. Каждый элемент этого поля представляет собой байт (8 бит), который можно интерпретировать как многочлен степени меньше 7:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2\ b_1x + b_0\ \#(1)$$

где $b_i \in \{0,1\}$ – коэффициент при соответствующей степени переменной x.

Арифметические операции в поле Галуа $GF(2^8)$ выполняются по модулю неприводимого многочлена. Неприводимый многочлен — это многочлен, который нельзя разложить на произведение двух многочленов меньшей степени с коэффициентами из того же поля. Другими словами, он аналогичен простому числу в арифметике целых чисел: его нельзя "разделить" на другие многочлены. В контексте $GF(2^8)$, неприводимый многочлен гарантирует, что любое умножение многочленов в поле может быть приведено к многочлену степени меньше 7 путем деления на неприводимый многочлен, представленный в формуле (1), и взятия остатка. Это свойство обеспечивает замкнутость поля, то есть все операции остаются внутри множества элементов поля

$$m(x) = x^8 + x^4 + x^3 + x + 1\ \#(2)$$

где m(x) – неприводимый многочлен для конечного поля Галуа $GF(2^8)$.

Сложение в $GF(2^8)$ выполняется по следующему алгоритму:

– каждый байт интерпретируется как многочлен степени меньше 7 с коэффициентами из множества {0,1}.

– выполняется сложение коэффициентов по модулю 2.

– результатом является новый многочлен, соответствующий байту, который также принадлежит полю GF(2^8).

Умножение в поле Галуа выполняется как умножение многочленов с последующим приведением результата по модулю неприводимого многочлена m(x).

Процесс умножения можно описать следующим образом:

– Каждый байт интерпретируется как многочлен степени меньше 7 с коэффициентами из множества {0,1}.

– Многочлены перемножаются по правилам обычного умножения многочленов, где коэффициенты складываются по модулю 2.

– Если результат имеет степень больше 8, он делится на m(x), и берется остаток от деления.

– Полученный остаток представляет собой новый многочлен, соответствующий байту, который также принадлежит полю GF(2^8).

1.2 Процесс шифрования

AES является итеративным блочным шифром, то есть процесс шифрования состоит из нескольких повторяющихся раундов. Каждый раунд включает последовательность преобразований, которые обеспечивают диффузию и перемешивание данных.

В алгоритме шифрования AES исходные данные разделяются на блоки по 128 бит, которые преобразуются в матрицы state размерностью 4×4 байта. Каждая матрица обрабатывается индивидуально с использованием четырёх ключевых методов преобразования: SubBytes, ShiftRows, MixColumns и AddRoundKey. Процесс шифрования включает три этапа: начальный, основной и завершающий.

На начальном этапе применяется метод преобразования AddRoundKey. AddRoundKey – Обеспечивает связь между шифруемыми данными и секретным ключом. Метод использует побитовое исключающее ИЛИ (XOR). Для выполнения AddRoundKey исходный блок данных и раундовый ключ представляются в

виде матриц 4×4 байт. Каждый байт матрицы состояния (State) поэлементно XOR-ится с соответствующим байтом раундового ключа. Операция AddRoundKey может быть описана следующим образом:

$$S'[i][j] = S[i][j] \oplus K[i][j] \#(3)$$

где S′[i][j] — новое значение байта матрицы в позиции i,j;

S[i][j] — исходное значение байта матрицы в позиции i,j;

K[i][j] — значение байта в позиции i,j первого раундового ключа;

$\oplus$ — побитовая операция XOR.

Основной этап реализуется через последовательность повторяющихся циклов, называемых раундами. Каждый раунд представляет собой сложную комбинацию криптографических преобразований, направленных на поэтапное «запутывание» исходных данных и их глубокую взаимосвязь с секретным ключом. Количество выполняемых раундов зависит от параметра Nr и равно Nr – 1. Каждый раунд включает поочередного выполнение следующих методов преобразования: SubBytes, ShiftRows, MixColumns и AddRoundKey.

Метод SubBytes в алгоритме AES представляет собой замену байтов, которая выполняется с использованием фиксированной таблицы подстановки, называемой S-блоком (Substitution Box). S-блок— это заранее определенная таблица размером 16×16 байт, содержащая все возможные значения байта. Каждый элемент таблицы вычисляется на основе сложной математической операции, которая включает нахождение обратного элемента в конечном поле GF(2^8) и применение аффинного преобразования. Эта операция применяется к каждому байту матрицы состояния. Байт представляется в виде двух шестнадцатеричных чисел s= (x, y), где x определяется четырьмя старшими битами байта, а y — четырьмя младшими. В таблице S-блок новое значение s′ выбирается на пересечении строки x и столбца y. Операция SubBytes может быть описана следующим образом:

$$s = \ S[i][j]$$

$$s' = S_{box}[x][y], \text{где } x = \left[\frac{s}{16}\right], y = s \ mod \ 16 \ \#(4)$$

Где s – исходный байт;

s` – новое значение байта;

S_{box} – фиксированная таблица;

x – старшие 4 бита байта s, номер строки;

y – младшие 4 бита байта s, номер столбца.

Метод ShiftRows выполняет циклический сдвиг байтов в каждой строке матрицы на различное количество позиций влево, что способствует распространению информации между столбцами. В процессе выполнения операции первая строка матрицы остаётся без изменений, вторая строка подвергается циклическому сдвигу на одну позицию влево, третья строка — на две позиции влево, а четвёртая строка — на три позиции влево.

Метод MixColumns выполняет линейное преобразование над каждым столбцом состояния, обеспечивая диффузию данных между столбцами матрицы состояния. Каждый столбец матрицы состояния (4 байта) умножается на фиксированную матрицу M, используя арифметику конечного поля Галуа GF(2^8). Операция MixColumns может быть описана следующим образом:

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}; \ S_j = \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}; \#(5)$$

$$S'_j = \ M * S_j$$

где M – фиксированная матрица, где каждый элемент представлен в виде байта в 16-ом системе исчисления;

S_j – j-тый столбец исходной матрицы;

S'_j- j-тый столбец новой матрицы.

Метод AddRoundKey идентична алгоритму в начальном этапе. Однако ключевое отличие заключается в том, что на каждом раунде используется уни-

кальный ключ, сгенерированный из исходного ключа через алгоритм расширения ключ.

Финальный этап представляет собой один раунд, аналогичный основному, но без этапа MixColumns.

Процесс расширения ключа в AES является критически важным этапом алгоритма, преобразующим исходный ключ шифрования (длиной 128, 192 или 256 бит) в последовательность раундовых ключей, используемых на каждом этапе шифрования.

Исходный ключ разбивается на Nk слов длиной 32 бита. Эти слова становятся первыми элементами массива раундовых ключей $W[0], W[1], \ldots, W[Nk-1]$. Дальнейшие слова генерируются рекурсивно:

$$\mathrm{W[i]} = W[i - Nk] \oplus F(W[i-1]);\ \mathrm{i} \geq \mathrm{Nk} \quad \#(6)$$

где F — функция преобразования;

W[i] – i-тое 32 битные слово расширенного ключа;

Nk – число слов в исходном ключе;

$\oplus$– побитовый операция XOR.

Функция преобразования выполняется в три этапа: сначала, если индекс i кратен Nk , слово W[i−1] подвергается циклическому сдвигу байтов влево (RotWord), затем каждый байт полученного слова заменяется через таблицу замен S-box (SubWord), аналогично операции SubBytes в основном шифровании, после чего результат объединяется через XOR с колонкой матрицы Rcon , номер которой определяется как i/Nk. Если i не кратен Nk, никаких преобразований не выполняются.

Для AES-256 дополнительно применяется операция SubWord к словам с индексами, кратными 4, но не кратными Nk, что усиливает перемешивание данных при работе с длинными ключами.

1.3 Процесс дешифровки

Дешифровка данных, зашифрованных с использованием стандарта AES, представляет собой обратный процесс по отношению к шифрованию. Однако её

реализация не является простым зеркальным отражением этапов шифрования: для корректного восстановления исходного текста требуется применение обратных преобразований и корректировка порядка операций.

Применение обратных преобразований заключается в необходимости инвертирования преобразования, использованных в методах SubBytes, ShiftRows и MixColumns. При дешифровке в методе обратная замена байтов (InvSubBytes) в отличии от шифрования используется другая матрица InvS-Box. В обратном сдвиге строк (InvShiftRows) происходит сдвиг строк, как и при шифровании только перемещение происходит в противоположном направлении, то есть в правую сторону. В обратном перемешивании столбцов (InvMixColumns) каждый столбец умножается на обратную фиксированную матрицу:

$$InvM = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \#(7)$$

где InvM – обратная фиксированная матрицы, используемая в методе InvMixColumns

Корректировка порядка операций предполагает изменение последовательности выполнения преобразующих методов на основном и финальном этапах алгоритма. Методы применяются в следующем порядке: InvShiftRows, InvSubBytes, AddRoundKey и InvMixColumns. На финальном этапе расшифрования, аналогично процессу шифрования, метод InvMixColumns не используется.

Ещё одним ключевым отличием дешифровки является порядок использования раундовых ключей. В шифровании ключи добавляются последовательно: AddRoundKey применяется сначала с начальным ключом, затем с первым раундовым ключом, и так далее. При дешифровке раундовые ключи используются в обратном порядке: последний ключ, сгенерированный на этапе расширения, применяется первым, а начальный ключ — в завершающем раунде.

2: АНАЛИЗ И ВЫБОР ПРОГРАММНЫХ СРЕДСТВ

Разработка мобильного приложения требует тщательного анализа и выбора программных средств, обеспечивающих надежность, безопасность и удобство использования. В данном разделе рассмотрены ключевые технологии, используемые в проекте, а также обоснование их выбора.

2.1 Язык программирования

Для разработки приложения выбран язык Kotlin. Kotlin — это современный статически типизированный язык программирования, разработанный компанией JetBrains в 2011 году как усовершенствованная альтернатива Java, а в 2017 году Kotlin получил официальную поддержку Google для разработки Android-приложений, что способствовало его широкому внедрению. Язык обладает следующими преимуществами:

– Официальная поддержка Google. Регулярные обновления, интеграция с Android Studio и документация от Google обеспечивают стабильность и актуальность языка.

− Совместимость с Java. Kotlin полностью совместим с Java, что позволяет использовать существующие библиотеки и API без необходимости их модификации. Это особенно важно при работе с устаревшим кодом или при интеграции с сервисами, предоставляющими Java SDK.

− Производительность. Компиляция в байт-код JVM гарантирует скорость, сопоставимую с Java

− Лаконичный и безопасный синтаксис. Код на Kotlin короче и выразительнее, чем аналогичный код на Java. Это снижает вероятность ошибок, упрощает сопровождение и делает разработку более продуктивной.

− Совместимость с Android SDK. Возможность использовать существующие библиотеки и инструменты экосистемы Android.

Kotlin демонстрирует высокую эффективность в разработке Android-приложений благодаря сочетанию лаконичного синтаксиса, безопасности, поддержки асинхронности и интеграции с современными фреймворками. Его выбор для дипломного проекта обусловлен официальной поддержкой Google, снижением трудозатрат на разработку и возможностью создания высокопроизводительных, отказоустойчивых приложений.

2.2 Среда разработки

Для создания приложения используется среда разработки (IDE) Android Studio. Android Studio, официальная интегрированная среда разработки для создания Android-приложений, разработанная Google на основе IntelliJ IDEA, объединяет множество инструментов, упрощающих создание, тестирование и оптимизацию приложений. Основной язык разработки — Kotlin, но поддерживаются также Java и C++ (для работы с NDK). Редактор кода предлагает автодополнение, рефакторинг, мгновенный анализ ошибок и интеграцию с инструментом Lint для проверки качества кода.

Визуальный конструктор интерфейсов позволяет разработчикам собирать пользовательские интерфейсы через drag-and-drop, используя XML для автоматической генерации кода. Встроенный ConstraintLayout упрощает создание адаптивных макетов за счет привязки элементов к краям контейнера или другим виджетам, а предпросмотр в режимах Design и Blueprint помогает оценить внешний вид на разных устройствах.

Эмулятор Android (AVD) имитирует работу реальных устройств, включая смартфоны, планшеты и носимые гаджеты, с поддержкой различных версий ОС, разрешений экранов и датчиков. Разработчики могут тестировать геолокацию, акселерометр, NFC и даже мультитач-жесты через виртуальную панель. Снимки состояния (Snapshots) ускоряют запуск эмулятора, а интеграция Google Play позволяет тестировать приложения с использованием сервисов компании.

Система сборки Gradle автоматизирует управление зависимостями, подключая библиотеки через файлы build.gradle, и поддерживает создание разных

версий приложения. Инструмент Build Analyzer помогает оптимизировать скорость сборки, выявляя медленные задачи.

Для анализа производительности Profiler отслеживает загрузку CPU, использование памяти, сетевой трафик и энергопотребление. CPU Profiler записывает трейсы методов для выявления узких мест, Memory Profiler анализирует утечки памяти через heap dumps, а Network Profiler логирует запросы с деталями URL и ответов. Energy Profiler оценивает влияние приложения на заряд батареи.

Logcat выводит логи в реальном времени с возможностью фильтрации по уровням (error, debug), тегам или тексту, а также интегрируется с ADB для выполнения команд. Database Inspector позволяет просматривать и редактировать данные SQLite, выполнять SQL-запросы и анализировать работу с Room.

Android Studio является оптимальной средой для разработки Android-приложений на Kotlin благодаря глубокой интеграции с Android SDK, встроенному эмулятору, поддержке Jetpack Compose, нативным инструментам для Kotlin и готовым шаблонам. Альтернативы уступают ей в следующих аспектах:

− IntelliJ IDEA – требует ручной настройки Android-плагинов, эмулятора и SDK, а также лишена специализированных инструментов, таких как визуальный редактор Jetpack Compose.

− VS Code – не имеет встроенных шаблонов проектов, эмулятора и полноценной поддержки XML-разметки без дополнительных плагинов, что усложняет разработку.

− Eclipse – обладает устаревшей архитектурой, сложной настройкой Android SDK и отсутствием поддержки современных технологий вроде Jetpack Compose.

Android Studio, несмотря на высокие требования к ресурсам, обеспечивает полный набор функций для профессиональной разработки. Альтернативные среды могут использоваться в узких сценариях, но не заменяют её функциональность

2.3 Хранение данных

В качестве базы данных выбрана SQLite – легковесная реляционная СУБД являющаяся одной из самых популярных инструментов для локального хранения данных в мобильных приложениях, благодаря своей компактности, простоте интеграции и эффективности в условиях ограниченных ресурсов мобильных устройств. Он входит в стандартные фреймворки Android что делает его естественным выбором для приложений, которым требуется работать офлайн, кэшировать или сохранять пользовательские данные. SQLite обеспечивает ACID-совместимость, гарантируя целостность данных даже при внезапных сбоях, а его встроенная архитектура, не требующая настройки сервера, и хранение всей базы в одном файле упрощают интеграцию и резервное копирование.

Ключевыми преимуществами SQLite в мобильной разработке являются низкие требования к ресурсам, скорость локальных операций чтения/записи и гибкость в управлении схемой базы. SQLite обладает возможностью интегрироваться с ORM-библиотеками, которые абстрагируют работу с SQL и снижают риск ошибок.

Однако у SQLite есть и ограничения. При частых одновременных записях возможны блокировки базы, что требует аккуратного управления потоками, особенно на старых версиях Android. Сложные JOIN-запросы или работа с большими объёмами данных могут замедлить приложение.

Учитывая ключевые требования проекта, такие как необходимость локального хранения данных и обеспечение высокой скорости операций чтения и записи, система управления базами данных SQLite демонстрирует оптимальное соответствие заданным критериям. Эта встроенная СУБД, не требующая отдельного сервера или сложной настройки, идеально подходит для сценариев, где приоритетом является простота развертывания и минимальное потребление ресурсов.

В качестве хранения внутренних данных и настроек приложения используется SharedPreferences.

SharedPreferences в Android – это встроенный механизм для хранения небольших данных в формате ключ-значение, предназначенный для сохранения настроек приложения, пользовательских предпочтений и временных параметров. Данные сохраняются в XML-файле в защищённой папке приложения, что обеспечивает их доступность после перезапуска.

К преимуществам относятся лёгкость использования, высокая скорость для малых объёмов информации, интеграция с Android SDK и надёжность сохранения данных через методы apply() (асинхронно) или commit() (синхронно).

К недостаткам можно отнести: отсутствие поддержки сложных структур или запросов, низкая производительность при работе с большими данными, отсутствие встроенной защиты чувствительной информации (например, паролей)

2.4 Система контроля версий

Для контроля версий и управления кодом используется GitHub. GitHub, основанная на распределённой системе контроля версий Git, является ключевым инструментом в современной разработке программного обеспечения. В контексте выполнения дипломных работ её функциональные возможности позволяют эффективно организовать процесс разработки, обеспечить сохранность данных и документирование этапов разработки. GitHub предоставляет облачное хранилище для исходного кода, что решает проблему резервного копирования. В случае технических сбоев или утраты локальных данных все версии проекта сохраняются в удалённом репозитории. Доступ к репозиторию возможен из любой точки мира через веб-интерфейс или специализированные приложения

Система контроля версий позволяет фиксировать изменения в коде или научных материалах с помощью коммитов, снабжённых временными метками и описаниями. Это даёт возможность анализировать историю правок для выявления ошибок или восстановления предыдущих версий, например, при возникновении конфликтов в данных. Встроенные инструменты сравнения обеспечивают детальную визуализацию различий между версиями файлов. Платформа автоматически выделяет изменения на уровне строк, включая добавления, удаления

и модификации, что позволяет точно отслеживать эволюцию кода или текстовых материалов. Эта функциональность упрощает анализ внесённых правок. Создание изолированных веток является одной из ключевых возможностей GitHub, обеспечивающих гибкость в управлении проектами. Каждая ветка представляет собой независимую линию разработки, что позволяет экспериментировать с новыми идеями, алгоритмами или архитектурными решениями без риска нарушить стабильность основной версии. После завершения работы над экспериментом изменения могут быть интегрированы в главную ветку через механизмы слияния (merge) или отклонены, если результат не соответствует ожиданиям. Такой подход минимизирует хаос в кодовой базе и способствует соблюдению принципов модульности.

3 ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ.

3.1 Методология разработки ПО

При разработке мобильного приложения для дипломной работы была выбрана методология Agile, ориентированная на итеративное развитие проекта, гибкость и постоянное улучшение. Основными принципами подхода являются фокус на пользовательские потребности, адаптивность к изменениям, разбиение задач на короткие циклы (спринты) и тестирование промежуточных результатов

Основная разработка велась по спринтам — коротким циклам фиксированной длительности, которые в данном случае составляли две недели. В рамках каждого спринта реализовывались конкретные модули приложения, определённые на этапе планирования в соответствии с приоритетами, сформированными на основе предыдущих итераций. После завершения спринта готовые функции тестировались с использованием симуляции реальных сценариев использования, что позволило выявлять ошибки и недочёты на ранних этапах, а также оперативно вносить корректировки в планы последующих итераций.

Каждый спринт включал анализ полученных результатов, что обеспечивало гибкость в перераспределении приоритетов задач. Например, если тестирование выявляло недостатки в интерфейсе, в следующем спринте повышался приоритет задач, связанных с улучшением удобством использования приложения. Такой подход соответствует принципам Agile, направленным на постоянное улучшение продукта через итеративное развитие и адаптацию к новым данным.

Кроме того, структура спринтов способствовала поддержанию регулярного ритма работы и фокусировки на краткосрочных целях, что особенно важно при самостоятельной разработке без внешнего заказчика. Это позволило сохранить прозрачность прогресса и минимизировать риски отклонения от ключевых целей проекта.

3.2 Анализ требований

Мобильное приложение для создания зашифрованных заметок предназначено для обеспечения безопасного хранения и управления конфиденциальной информацией в условиях роста киберугроз и необходимости защиты личных данных. Оно предоставляет пользователям инструменты для создания текстовых заметок, содержащих чувствительные данные — пароли, номера банковских карт, личные переписки, бизнес-планы, медицинские записи и другие сведения, требующие повышенной защиты.

Ключевая цель приложения — минимизировать риски несанкционированного доступа к данным за счет реализации локального шифрования на устройстве пользователя, исключающего передачу информации на сторонние серверы. Это делает его особенно актуальным для лиц, сталкивающихся с необходимостью хранения информации, недоступной для третьих сторон, таких как журналисты, юристы, представители финансовой сферы или обычные пользователи, заботящиеся о приватности.

Помимо обеспечения безопасности, приложение должно выполнять следующие задачи:

- Быстрый доступ к заметкам через интуитивный интерфейс.
- Импорт/экспорт заметок.
- Аутентификация пользователя .
- Автоматическая очистка памяти при сворачивании или блокировке устройства.

Для достижения поставленной цели и задач были выделенные следующие спринты:

- Спринт 1 – Подготовка программной среды и инструментов для разработки приложения;
- Спринт 2-3 – Реализация модуля шифрования;
- Спринт 3-4 – Создание интерфейса приложения;
- Спринт 4-6 – Реализация прикладной логики;

− Спринт 7 – Финальное тестирование.

3.3 Подготовка программной среды и инструментов для разработки приложения

Для успешного начала разработки мобильного приложения необходимо подготовить программную среду и инструменты, которые обеспечат эффективную работу с кодом, версионный контроль и интеграцию всех компонентов проекта. Первым шагом становится настройка системы управления версиями Git с использованием платформы GitHub. Регистрация на GitHub проводится на официальном сайте. Для этого потребуется указать адрес электронной почты и придумать надежный пароль. После подтверждения email-адреса аккаунт будет активирован, и можно приступить к созданию первого репозитория.

Репозиторий на GitHub служит центральным хранилищем для всех файлов проекта, включая код, документацию и конфигурационные данные. Чтобы создать репозиторий, нужно нажать кнопку «New repository» в личном кабинете. В открывшейся форме следует указать имя проекта, краткое описание и выбрать режим доступа — публичный или приватный. Публичный репозиторий виден всем, а приватный ограничивает доступ только для владельца и приглашенных участников. Также рекомендуется активировать опции «Add a README file» для начальной документации, «Add .gitignore» для исключения ненужных файлов из версионного контроля и «Choose a license» для указания прав использования кода. Эти файлы автоматически генерируются системой, упрощая начало работы и устанавливая стандарты управления проектом.

Следующим этапом подготовки становится установка программы GitHub Desktop, которая предоставляет графический интерфейс для работы с Git, упрощая операции, такие как коммиты, ветвление и синхронизация. Для загрузки приложения нужно перейти на страницу https://desktop.github.com/ , выбрать версию под операционную систему Windows и запустить установочный файл. В процессе установки следует следовать инструкциям мастера, приняв лицензионное соглашение и указав папку для сохранения данных. После завершения

установки программа предложит войти в ранее созданный аккаунт GitHub, что свяжет локальные репозитории с облачным хранилищем. GitHub Desktop особенно удобен, так как визуализирует процессы управления версиями.

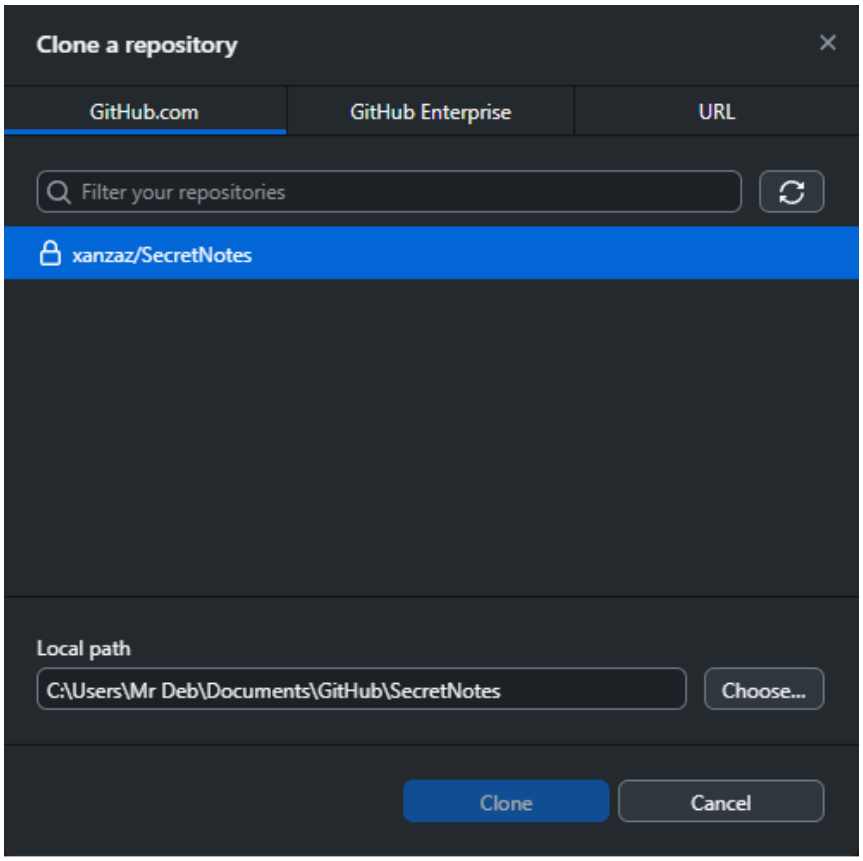

Рисунок 1 – Окно клонирования репозитория GitHub

После установки GitHub Desktop и авторизации в аккаунте пользователь приступает к интеграции локального рабочего пространства с облачным репозиторием. Для начала требуется клонировать удалённый проект с GitHub на компьютер, чтобы начать работать с кодом локально. В интерфейсе GitHub Desktop выбирается опция «Clone a repository. После этого программа автоматически открывает окно со списком всех репозиториев пользователя на GitHub (рисунок 1). Современная версия GitHub Desktop упрощает процесс: вместо ручного копирования URL достаточно выбрать нужный проект из выпадающего списка и указать путь к локальной папке. После подтверждения настроек GitHub Desktop запускает процесс клонирования, которая автоматически скачи-

вает все файлы из облачного хранилища в указанную папку и синхронизирует их с локальным репозиторием. Теперь локальный репозиторий полностью синхронизирован с облачным. Любые изменения в коде можно фиксировать локально через кнопку Commit to main , а затем отправлять их в GitHub с помощью Push origin.

Следующим шагом является установка и настройка среды разработки. Как и в предыдущем шаге скачиваем с официального сайта установочный файл Android Studio и проходим процесс установки.

После установки приступаем к созданию проекта. Запускаем функцию «New project» и в открывшемся окне выбираем шаблон приложения «Empty Activity». Далее задаем название проекта, выбираем директорию, которую создали на этапе настройки GitHub, выбираем язык программирования Kotlin. минимальную версию Android поддерживаемая приложением выбираем API 24, так как созданное приложение будет поддерживаться практически на всех Android устройствах. А систему сборки приложения (Bild configuration language) выбираем Kotlin DSL.

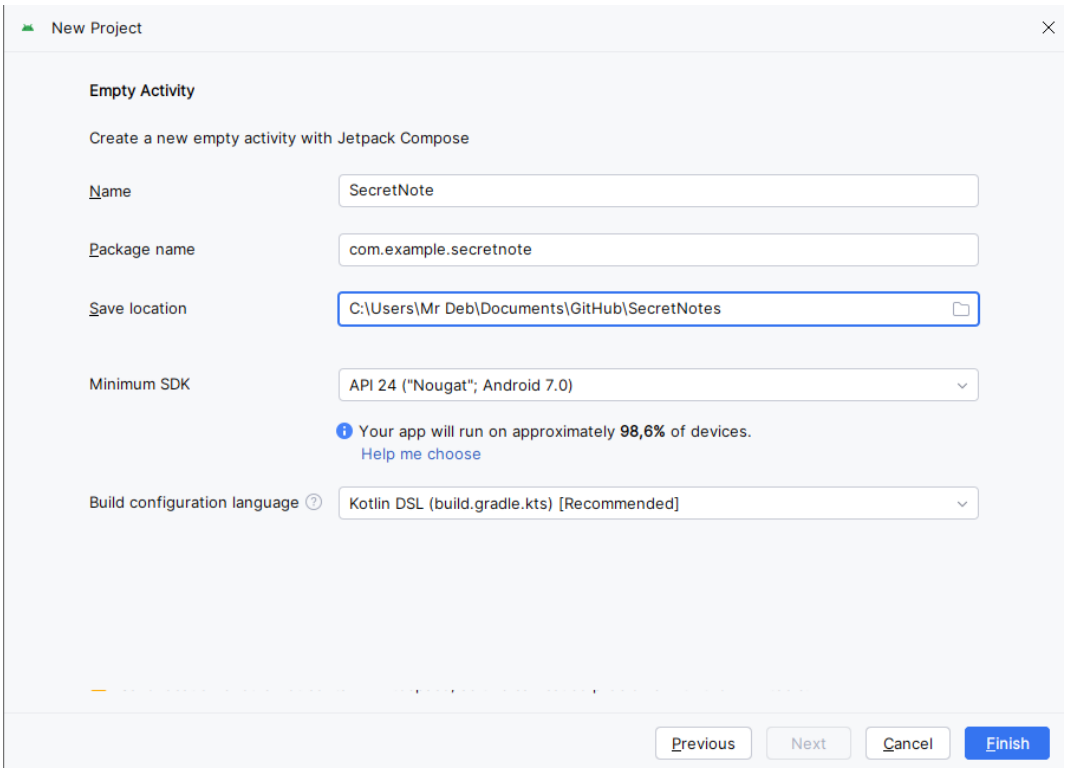

Рисунок 2 – Окно создания нового проекта в Android Studio

После создания проекта в Android Studio формируется иерархия файлов и папок, каждая из которых выполняет определенную роль в разработке, сборке и запуске приложения. На верхнем уровне находится корневая директория проекта, которая содержит глобальные настройки и конфигурации. Здесь расположен файл build.gradle, где указываются версия Gradle, репозитории и зависимости, необходимые для всей проектной среды. Рядом с ним находится settings.gradle, определяющий, какие модули включены в проект. Глобальные параметры, такие как версия JDK или настройки памяти Gradle, хранятся в gradle.properties, а индивидуальные пути к Android SDK указываются в local.properties.

Основная часть проекта сосредоточена в модуле app, который включает исходный код, ресурсы и конфигурации приложения. Внутри src/main/ находятся директории java/ или kotlin/ с классами Java/Kotlin, включая точку входа — MainActivity. Ресурсы интерфейса хранятся в res/, где папки layout/ содержат XML-файлы разметки, а values/ — строки, цвета, стили и размеры. Файлы изображений и векторной графики размещаются в drawable/ или mipmap/, а дополнительные данные, такие как аудио или JSON, — в raw/. Файл AndroidManifest.xml описывает ключевые параметры приложения: имя пакета, минимальную и целевую версии Android, разрешения и компоненты.

Для тестирования используются папки test/ и androidTest/, где хранятся локальные и интеграционные тесты соответственно. Модульные тесты на основе JUnit проверяют логику приложения, а интеграционные, например, с использованием Espresso, запускаются на эмуляторе или физическом устройстве. Конфигурация модуля app/ описывается в build.gradle, где указываются версии SDK, тип модуля и зависимости библиотек.

Финальным этапом настройки Android Studio является создание нескольких виртуальных устройств с разными параметрами — это позволяет провести более детальное и качественное тестирование приложений. Для этого используется Device Manager — инструмент, позволяющий симулировать работу приложения на различных моделях устройств и версиях Android. Для открытия Device

Manager следует выбрать в меню IDE. На открывшейся вкладке (рисунок 3) отображается список ранее созданных виртуальных устройств. Для добавления нового устройства нажимается кнопка «Create Virtual Device».

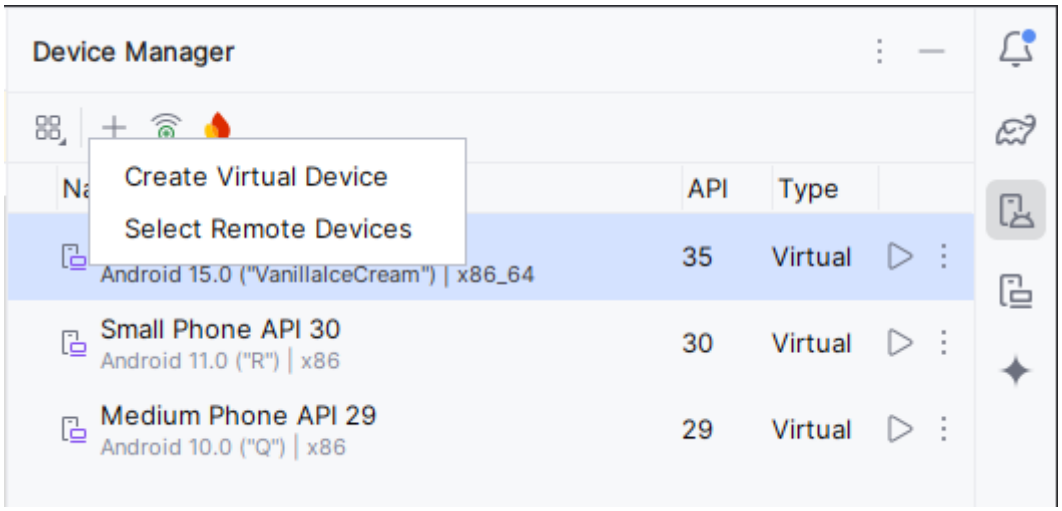

Рисунок 3 – Окно Device Manager

На этапе создания виртуального устройства требуется выбирает категорию устройства и конкретную модель из предложенного списка (Рисунок 4).

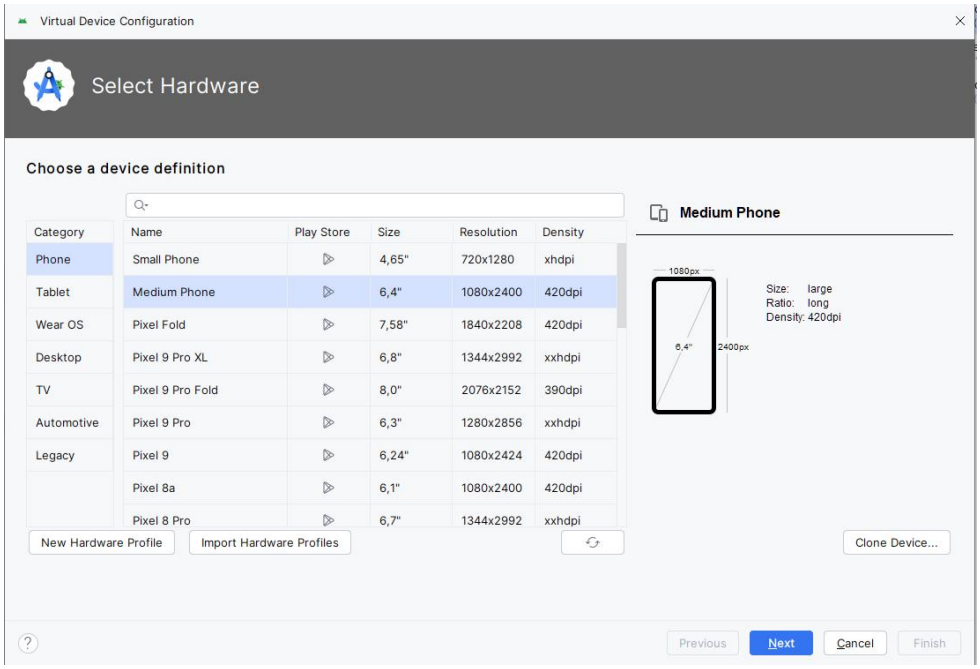

Рисунок 4 – Окно выбора модели создаваемого виртуального устройства

Далее система предлагает указать версию Android, которая будет установлена на эмуляторе (Рисунок 5).

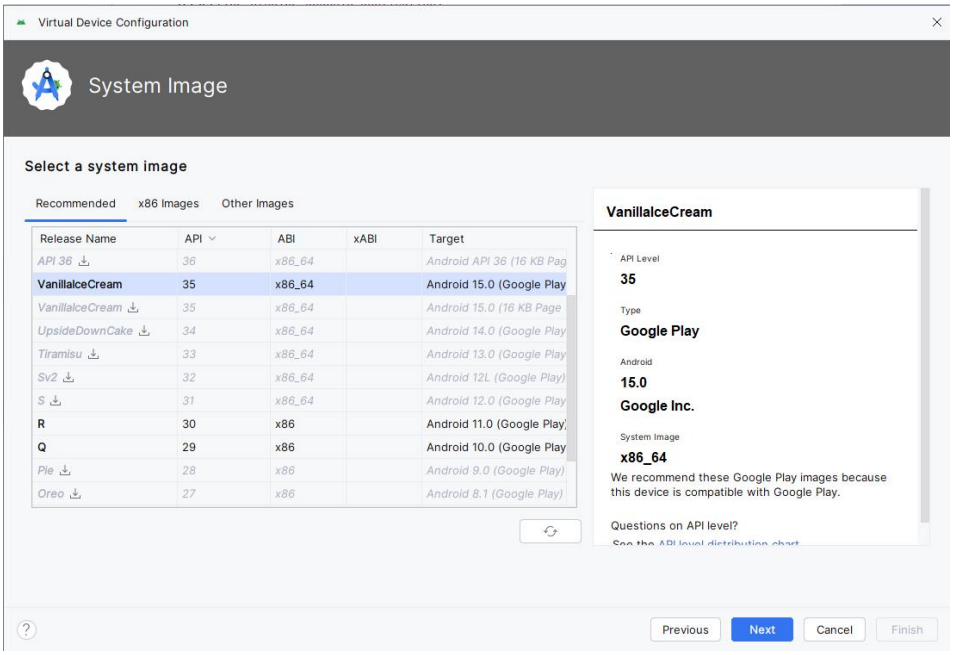

Рисунок 5 – Окно выбора версии Android

Если требуемая версия Android не установлена, Device Manager автоматически предложит загрузить её через SDK Manager.

3.4 Реализация модуля шифрования

Для обеспечения безопасности пользовательских данных в Android-приложении реализован модуль шифрования на основе стандарта AES (Advanced Encryption Standard). Модуль состоит из трех ключевых классов:

– Класс GF — реализация арифметики в конечных полях GF(2^8).

– Класс State — класс, представляющий данные в виде матрицы 4×4 и реализующий этапы преобразований AES.

– Класс AES — основной класс, управляющий генерацией ключей, разбиением сообщений на блоки и процессом шифрования/дешифрования.

Для реализации операций в конечном поле GF(2^8) был разработан специализированный класс GF, который инкапсулирует методы для выполнения сложения и умножения элементов. Основная задача класса — обеспечить эффек-

тивное выполнение математических операций, необходимых для алгоритма шифрования. В Kotlin перегрузка операторов реализуется через ключевое слово operator: для сложения используется метод plus(), а для умножения — times(). Это позволяет применять привычные символы + и * при работе с элементами $GF(2^8)$, сохраняя читаемость кода.

Сложение в $GF(2^8)$ выполняется через побитовое XOR. Гораздо более сложной является реализация умножения. Функция gfMultiply(a: Int, b: Int) реализует умножение двух элементов в конечном поле $GF(2^8)$. Алгоритм основан на бинарном разложении множителя, последовательных сдвигах и операции XOR, что позволяет избежать сложных операций с полиномами.

В начале функции инициализируются переменные: p хранит промежуточный результат умножения, а aVar и bVar получают значения входных параметров a и b. Далее выполняется цикл, повторяющийся 8 раз — по числу битов в 8-битном числе. На каждой итерации проверяется младший бит bVar. Если он равен 1, текущее значение aVar добавляется к результату p через операцию XOR.

После этого значение переменной aVar удваивается путём сдвига влево на один бит. Такой сдвиг эквивалентен умножению соответствующего полинома на многочлен x в рамках полиномиального представления. Однако если старший бит aVar до сдвига был равен 1, что при сдвиге приведёт к выходу за пределы 8-битного диапазона, выполняется коррекция через XOR с неприводимым полиномом 0x1B. Это действие гарантирует, что результат умножения останется в пределах $GF(2^8)$, Завершающим этапом текущей итерации алгоритма становится выполнение сдвига вправо на один бит для переменной- bVar. Такой сдвиг реализует переход к обработке следующего бита на последующей итерации.

После завершения всех итераций функция возвращает результат p, ограниченный маской 0xFF, чтобы оставить только младшие 8 битов. Этот шаг необходим для корректного представления чисел в диапазоне 0–255. Таким образом, алгоритм моделирует умножение в $GF(2^8)$ через последовательные битовые операции, избегая явного вычисления полиномиального произведения. Его эф-

фективность заключается в использовании сдвигов и XOR вместо сложных арифметических действий, что делает его подходящим для низкоуровневых реализаций. Реализация метода gfMultiply на языке Kotlin представлена на рисунке 6

```kotlin
private fun gfMultiply(a: Int, b: Int): Int {
    var p = 0 // Результат умножения
    var aVar = a
    var bVar = b

    for (i in 0 ≤ until < 8) {
        if (bVar and 1 != 0) { // Если младший бит bVar равен 1
            p = p xor aVar // Добавляем aVar к результату
        }
        val carry = aVar and 0x80 // Проверяем старший бит aVar
        aVar = aVar shl 1 // Сдвигаем aVar влево на 1 бит
        if (carry != 0) { // Если был перенос
            aVar = aVar xor 0x1B // Выполняем XOR с неприводимым полиномом
        }
        bVar = bVar shr 1 // Сдвигаем bVar вправо на 1 бит
    }

    return p and 0xFF // Ограничиваем результат 8 битами
}
```

Рисунок 6 – Реализация метода умножения полей Галуа

В AES умножение в GF(2^8) применяется на этапах MixColumns и InverseMixColumns. В MixColumns столбцы матрицы состояния умножаются на коэффициенты 1, 2, 3, а в InverseMixColumns — на числа 9, 11, 13, 14. Таким образом, все вычисления ограничиваются строго определёнными множителями: 2, 3, 9, 11, 13 и 14. Для проверки корректности реализации функции gfMultiply использовались эталонные таблицы умножения, заранее вычисленные для каждого из указанных чисел. Тестирование проходило следующим образом:

– Для каждого числа из списка создавалась таблица, где каждому возможному входному байту (0–255) соответствовал заранее рассчитанный результат умножения в GF(2^8).

– Функция gfMultiply вызывалась для всех 256 значений входного байта, умножая их на одно из чисел из списка.

– Полученные результаты сравнивались с эталонной таблицей.

Полное совпадение всех 256 вариантов для каждого числа подтвердило точность реализации.

Для реализации операций преобразований над состоянием данных был разработан класс State. В контексте алгоритма AES состояние данных представляет собой ключевую структуру, которая подвергается многочисленным преобразованиям на этапах шифрования и дешифрования. Состояние организовано как двумерный массив байтов размером 4×4, где каждый элемент соответствует одному байту входного блока данных. Этот массив называют State, и его трансформация осуществляется через последовательность операций. В рамках объектно-ориентированного подхода к реализации AES, класс State инкапсулирует эти основные преобразования, обеспечивая модульность и удобство тестирования отдельных компонентов алгоритма.

Класс State реализует ключевые преобразования AES – SubBytes, ShiftRows, MixColumns и AddRoundKey, которые инкапсулированы как приватные методы для исключения некорректного использования и обеспечения строгой последовательности выполнения операций. Эти методы недоступны для внешнего вызова напрямую, так как их порядок применения строго регламентирован стандартом AES. Вместо этого взаимодействие с классом осуществляется через публичные методы encoding() и decrypt(), которые реализуют полный цикл шифрования и дешифрования соответственно.

Метод encoding() управляет применением приватных преобразований в соответствии с алгоритмом AES: сначала выполняется начальный этап AddRoundKey, затем — итеративное применение раундовых операций, включая финальный раунд без MixColumns, как описано в разделе 1.2. Аналогично, метод decrypt() реализует обратные преобразования, строго соблюдая порядок действий для корректного восстановления данных.

Такая организация кода гарантирует, что все внутренние преобразования будут выполнены корректно, без риска нарушения логики алгоритма из-за сторонних вызовов. Приватные методы остаются изолированными, а внешние компоненты взаимодействуют с State только через строго определенные публичные интерфейсы — encoding() и decrypt(). Это усиливает безопасность, упрощает тестирование и позволяет легко интегрировать класс в общий процесс шифрования/дешифрования.

```
fun encoding(key: List<List<UByte>>)
{
    //Начальный этап шифрования
    AddRoundKey(key)
    //Основные раунды шифрования
    for (rnd in 1 ≤ until < Nr){
        SubBytes()
        ShiftRows()
        MixColums()
        AddRoundKey(key,rnd)
    }
    //Финальный этап шифрования
    SubBytes()
    ShiftRows()
    AddRoundKey(key,Nr)
}
```

```
fun decrypt(key: List<List<UByte>>)
{
    //Начальный этап шифрования
    AddRoundKey(key,Nr)
    //Основные раунды шифрования
    var rnd:Int = Nr - 1
    while (rnd >=1)
    {
        ShiftRows(inv = true)
        SubBytes(inv = true)
        AddRoundKey(key,rnd)
        MixColums(inv = true)

        rnd -=1
    }

    //Финальный этап шифрования
    ShiftRows(inv = true)
    SubBytes(inv = true)
    AddRoundKey(key,rnd)
}
```

Рисунок 7 – Код шифрования и дешифрования матрицы состояния

Класс AES представляет собой центральный элемент программной реализации алгоритма шифрования, который объединяет все ключевые компоненты: классы Галуа для выполнения арифметических операций, State для представления обрабатываемых данных и метод генерации раундовых ключей. Его основная задача — координировать работу всех модулей и шифрования и дешифрования текста.

Генерация ключей шифрования реализована в методе transformTextToArrKey, входными данными является пароль пользователя. Рас-

ширение исходного ключа происходит по принципу, описанному в главе 2.2 «Принципы работы AES», где приведены этапы преобразования ключей с использованием операций циклического сдвига байтов в слове, замена байтов через таблицу S-box и XOR с предвычисленной таблицей Rcon.

Шифрование текста реализуется через метод encodingText, который принимает на вход данные, подлежащие преобразованию. Перед началом обработки входной текст дополняется до длины, кратной 16 байтам — это необходимо для корректного разбиения на блоки, соответствующие стандартному размеру, используемому в блочном шифровании.

После дополнения текст разделяется на фрагменты по 16 байт каждый. Эти блоки последовательно преобразуются в объекты класса State, формируя массив. Для каждого объекта из созданного массива вызывается метод encoding, выполняющий непосредственное шифрование. Результат выполнения этого метода для каждого блока сохраняется в отдельной переменной. После шифрования всех объектов State массива полученные данные объединяются в единую переменную, которая содержит финальный результат шифрования исходного текста.

Метод дешифрования decryptText реализован аналогично процессу шифрования, но с ключевым отличием: вместо метода encoding для объектов класса State вызывается метод decrypt. Входным параметром для метода дешифрования служит массив байтов, представляющий зашифрованные данные.

3.5 Проектирование базы данных

Проектирование базы данных – это многоэтапный процесс создания структуры хранения данных, который обеспечивает их целостность, эффективность и соответствие требованиям приложения. Проектирование делиться на три уровня: инфологическое, логическое и физическое проектирование. Каждый этап решает свою задачу и подготавливает данные для реализации в конкретной СУБД.

Инфологическое проектирование — это этап анализа предметной области, на котором выявляются сущности, их атрибуты и логические связи между ними. На этом этапе создаётся абстрактная модель данных, не зависящая от конкретной СУБД.

После анализа создаваемого приложения были выделены две сущности:

- Сущность «Заметки», хранящая в себе все обычные заметки.
- Сущность «Зашифрованные заметки», хранящая в себе все зашифрованные заметки.

Спецификация атрибутов представлена в таблице 2–3.

Выше представленные сущности автономны и не имеют никаких связей. Из-за чего определение связей между сущностей в инфологическом проектировании не производиться.

Результатом инфологического проектирования является инфологическая модель, которая представлена на рисунке 8

Таблица 2 – Спецификация атрибутов сущности «Заметки»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример
Код заметки	Числовое обозначение, определяющее каждую заметку	Числовой	>0	34
Заголовок	Заголовок заметки	Текстовый		Повышение продуктивности с помощью матрицы Эйзенхауэра
Содержание	Текстовое содержание заметки	Текстовый		Разделите задачи на четыре категории: «срочные и важные…
Дата и время	Дата и время создания заметки	Дата и время	> текущей даты	05.03.2025 16:42:12

Таблица 3 – Спецификация атрибутов сущности «Зашифрованные заметки»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример
Код заметки	Числовое обозначение, определяющее каждую зашифрованную заметку	Числовой	>0	8
Заголовок	Заголовок зашифрованной заметки	Массив байтов		e3 a9 d2 f1 4c 7b 8a 3d 9e 5f 0d 7c 2a 1b 8e …
Содержание	Текстовое содержание зашифрованной заметки	Массив байтов		b7 a6 f5 e4 d3 c2 b1 a0 f9 e8 d7 c6 b5 a4 f3 e2 …
Дата и время	Дата и время создания зашифрованной заметки	Массив байтов		c4b3a2f1e0d9 c8b7a6f5e4d3 c2b1a0f9

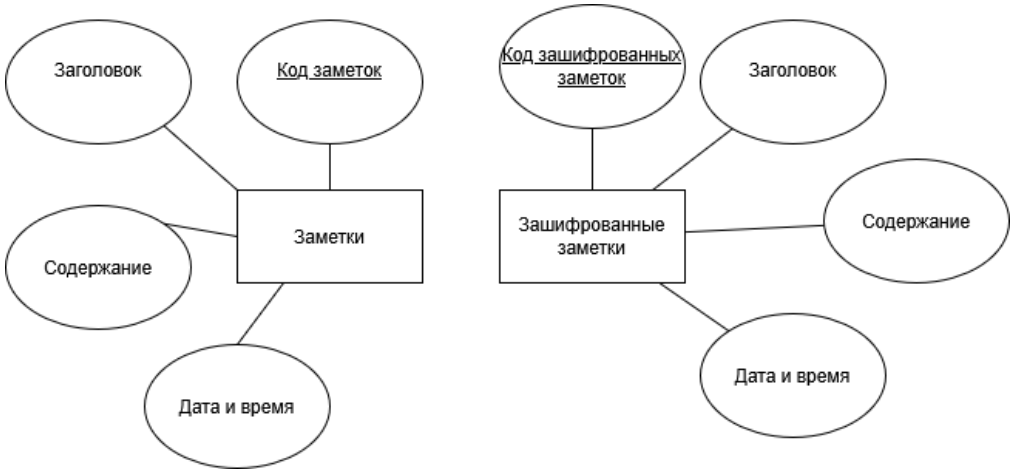

Рисунок 8 – Инфологическая модель в нотации Чена

Логическое проектирование — это этап преобразования концептуальной модели в структуру таблиц, типов данных, ключей и ограничений. Здесь определяется, как данные будут организованы в реляционной базе данных, но без привязки к конкретной СУБД. Процесс логического проектирования включает

несколько этапов: преобразование модели «Сущность-связь» и нормализация данных.

На этапе преобразовании модели «Сущность-связь» каждая сущность концептуальной модели преобразуется в таблицу, атрибуты сущности, описывающие её характеристики, становятся столбцами этой таблицы. Связи между сущностями реализуются через внешние ключи. Для отношений типа «один ко многим» внешний ключ добавляется в таблицу, соответствующую сущности, находящейся на стороне «многие». Это связывает записи через ссылку на первичный ключ связанной таблицы. Для отношений типа «многие-ко-многим» создается промежуточная таблица, содержащая внешние ключи, ссылающиеся на первичные ключи обеих связанных таблиц. Такой подход позволяет хранить сложные взаимосвязи без дублирования данных.

Однако в рассматриваемой базе данных отсутствуют связи между сущностями, что исключает необходимость реализации внешних ключей и промежуточных таблиц.

Нормализация данных в базе данных – это процесс организации данных в таблицах с целью устранения избыточности и повышения эффективности хранения и обработки данных. Он включает приведение таблиц к нормальным формам:

− Первая нормальная форма требует, чтобы каждый атрибут был атомарным, т.е атрибут должен хранить единственное значение и не являться ни списком, ни множеством значений.

− Вторая нормальная форма требует нахождение отношения в первой нормальной форме и каждый не ключевой атрибут должен зависеть от ключа.

− Третья нормальная форма требует нахождение отношения во второй нормальной форме и каждый не ключевой атрибут должен зависеть от ключа и не зависят друг от друга.

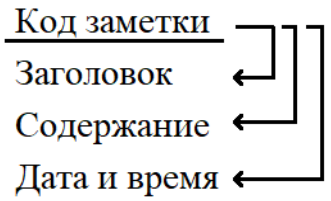

Рисунок 9 – Диаграмма функциональных зависимостей отношения «Заметки»

Отношение «Заметки» находится в третьей нормальной форме, т.к удовлетворяет всем требованиям. Отношение «Зашифрованные заметки» обладает идентичным набором атрибутов и структурой данных по сравнению с отношением «Заметки». Поскольку структурные и функциональные зависимости между атрибутами сохраняются, данное отношение также находится в третьей нормальной форме.

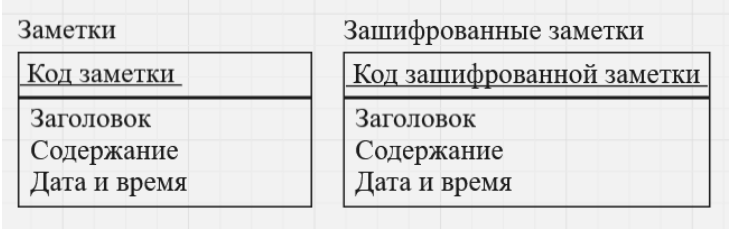

Рисунок 10 – Логическая модель базы данных – диаграмма IDEF1X

Физическое проектирование базы данных представляет собой завершающий этап проектирования базы данных, заключающийся в преобразовании логической модели данных в конкретную реализацию. Этот процесс направлен на определение физических структур хранения, таких как таблицы, столбцы, индексы, а также настройку параметров СУБД.

Таблица 4 – Физическая структура данных отношения «Заметки»

Название атрибута	Тип данных	Условия	Формат данных	Индексация
Код заметки	Числовой	> 0	INTEGER	Primary key, Autoincrement
Заголовок	Текстовый	-	TEXT	-
Содержание	Текстовый	-	TEXT	-
Дата и время	Текстовый	-	TEXT	-

Таблица 5 – Физическая структура данных отношения «Секретные заметки»

Название атрибута	Тип данных	Условия	Формат данных	Индексация
Код заметки	Числовой	> 0	INTEGER	Primary key, Autoincrement
Заголовок	Бинарные данные	-	BLOB	-
Содержание	Бинарные данные	-	BLOB	-
Дата и время	Бинарные данные	-	BLOB	-

3.6 Создание интерфейса

При разработке дизайна мобильного приложения основное внимание было уделено обеспечение высокой степени интуитивной понятности. Для снижения порога входа и ускорения адаптации пользователей принято решение использовать паттерны взаимодействия и визуальные шаблоны, заимствованные из предустановленных системных приложений заметок.

Цветовая палитра разработана на основе анализа предустановленных приложений. В светлой теме доминируют нейтральные тона для фона, обеспечивающие минимальную зрительную усталость, и яркие акцентные цвета для выделения ключевых элементов управления. Для темной темы используют глубокие черные или темно-серые оттенки, что снижает энергопотребление и минимизирует усталость глаз в условиях слабого освещения. Текст и иконки используют светло-серые оттенки, обеспечивая высокую читаемость. Акцентные элементы, такие как кнопки, сохраняют темно-фиолетовый цвет

Для разработки пользовательского интерфейса использовался Design Editor. Design Editor — это визуальный инструмент, позволяющий создавать и редактировать пользовательские интерфейсы приложений, автоматически генерируя XML-код разметки. Он интегрирован в IDE и предоставляет возможность работать с макетами через перетаскивание элементов, настройку свойств и предварительный просмотр. Основные компоненты интерфейса включают палитру с элементами UI, дерево компонентов для управления иерархией, панель

атрибутов для изменения свойств и рабочую область, где отображается холст с текущим экраном приложения.

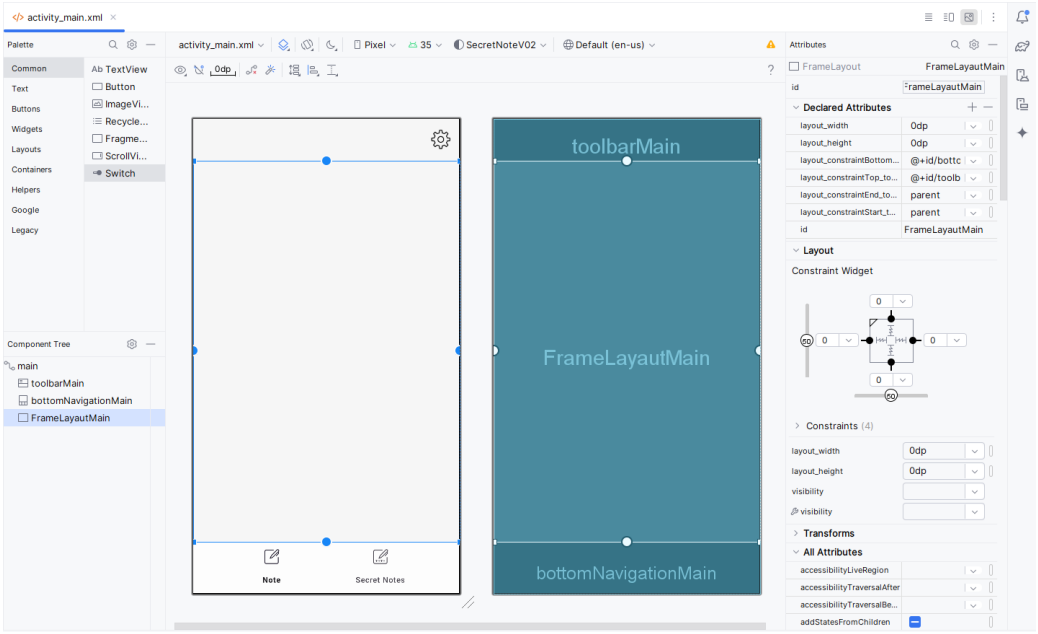

Рисунок 11 – Интерфейс Design Editor

Дизайн приложения строится на основе модульного подхода, который сочетает использование Activity и Fragments. Такая архитектура позволяет создавать гибкие и поддерживаемые интерфейсы, адаптированные под различные устройства и сценарии использования.

Activity выступает как контейнер верхнего уровня, представляющий отдельный экран приложения. Каждая Activity управляет своим жизненным циклом, обеспечивает отображение пользовательского интерфейса и взаимодействует с системными компонентами Android. Она служит основой для статичных элементов экрана, таких как навигационная панель или заголовок.

Fragments, в свою очередь, представляют собой модульные блоки интерфейса, которые можно динамически добавлять, удалять или заменять внутри Activity. Каждый фрагмент имеет собственный жизненный цикл и отвечает за реализацию конкретного функционала, например, отображение списка данных или формы ввода. Это позволяет разделять сложные экраны на независимые компоненты, упрощая их повторное использование и тестирование.

Основная цель такого подхода — разделить интерфейс на логически завершённые блоки. Это повышает гибкость разработки, упрощает внесение изменений и расширение функционала. Например, один и тот же фрагмент может использоваться в разных Activity или адаптироваться под различные размеры экранов.

Реализация экранов происходит следующим образом: Activity задаёт общий шаблон, включающий неизменяемые элементы интерфейса, а динамическое содержимое формируется через заменяемые фрагменты. Такой подход сохраняет визуальное единство приложения, одновременно позволяя обновлять отдельные части экрана без перезагрузки всей страницы.

Главный экран содержит неизменяемые элементы: верхнюю панель инструментов и нижнее меню. Основное содержимое отображается в динамическом контейнере для фрагментов. При выборе раздела «Заметки» или «Зашифрованные заметки» в контейнер загружается соответствующий фрагмент. Однако если пользователь не прошёл аутентификацию, при переходе в раздел «Зашифрованные заметки» автоматически подключается фрагмент авторизации. Это обеспечивает защиту конфиденциальных данных, блокируя доступ до подтверждения личности, при этом сохраняя целостность интерфейса.

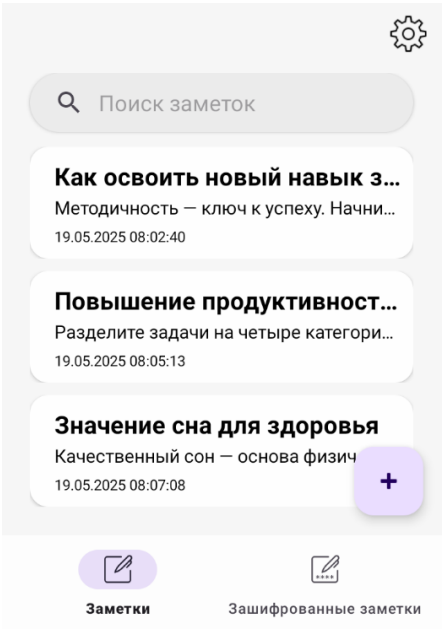

Рисунок 12 – Главное экран с фрагментом заметки

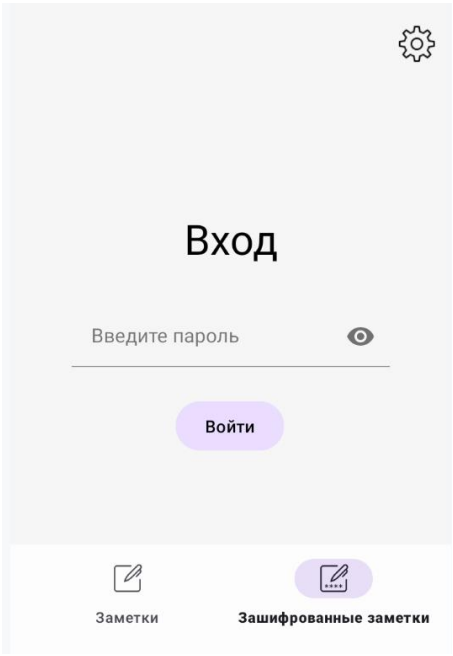

Рисунок 13 – Главное экран с фрагментом авторизации

При работе с заметками интерфейс предлагает не только базовую навигацию, но и удобное контекстное нижнее меню, активируемое при выделении одной или нескольких заметок. Оно предоставляет ключевые действия для работы со списком заметок. На рисунках 14 и 15 показаны два варианта меню: для обычных и зашифрованных заметок соответственно. Оба интерфейса включают две идентичные функции. Первая кнопка отвечает за выбор всех заметок в списке или очистку текущего выделения. Вторая позволяет удалить отмеченные заметки. Третья кнопка в каждом меню выполняет уникальные задачи: для обычных заметок (рис. 11) она предназначена для шифрования выделенных записей, превращая их в защищённые данные, а для зашифрованных заметок (рис. 12) обеспечивает импорт выбранных записей в файл

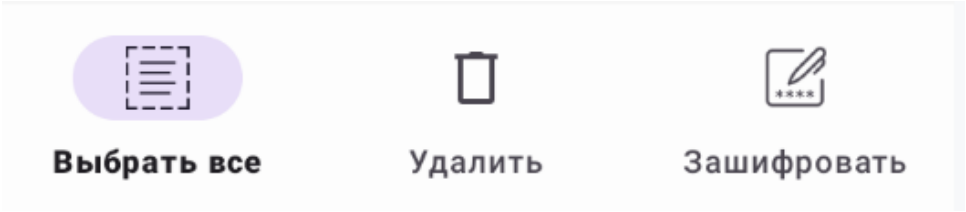

Рисунок 14 – Нижнее меню главного экрана при выделении обычных заметок

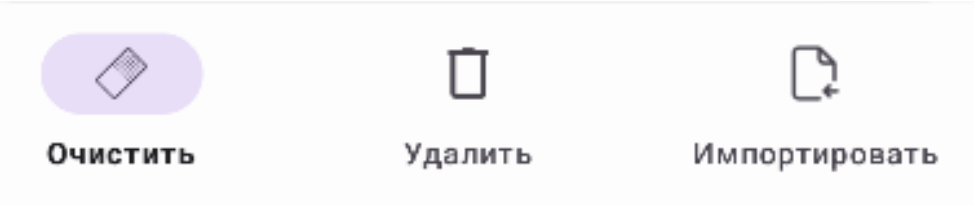

Рисунок 15 – Нижнее меню главного экрана при выделении зашифрованных заметок

Во фрагментах «Заметки» и «Зашифрованные заметки» реализована функция поиска, обеспечивающая оперативное нахождение записей, по ключевым словам, в заголовках и содержании. Отображение результатов поиска организовано через динамически обновляемый список карточек заметок. При обнаружении искомого фрагмента в содержании заметки текст карточки заменяется на фрагмент, включающий сам запрос и окружающий его контекст.

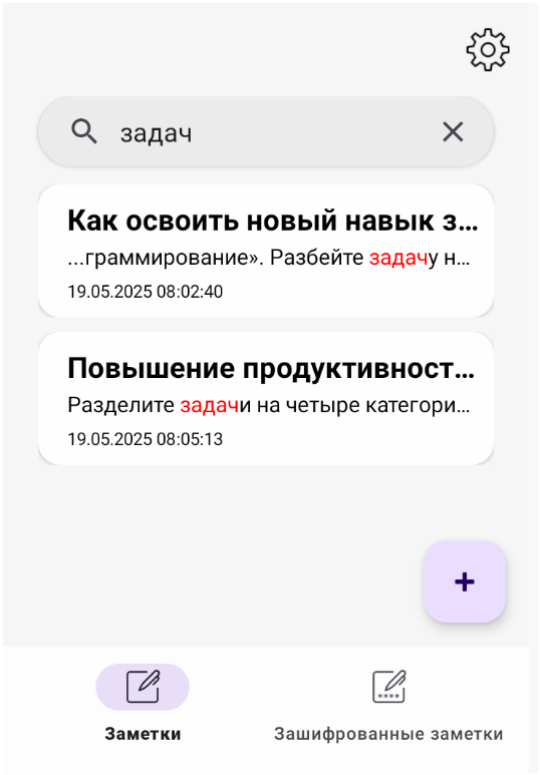

Рисунок 16 – Демонстрация работы поиска по заметкам

Экран создания и редактирования заметок предназначен для работы с отдельной заметкой и состоит из Activity с расширенным toolbar, включающим

иконки возврата и сохранения. Основная часть содержит два текстовых поля: для заголовка и содержимого заметки. Сохранение данных происходит через нажатие на соответствующую иконку в toolbar, что активирует проверку введённых данных и запись информации в хранилище.

Если заметка была открыта из списка результатов поиска, найденный фрагмент автоматически выделяется цветом. Кроме того, если фрагмент находится вне области видимости текстового поля, реализована автоматическая прокрутка к его позиции.

Рисунок 17 – Экран создания и редактирования заметок

Экран настроек состоит из toolbar, содержащий кнопку возврата, и контейнера фрагмента. В начальном состоянии контейнер отображает фрагмент со списком настроек: смена пароля, экспорт всех зашифрованных заметок, импорт заметок и выбор версии шифрования. При нажатии на элемент списка происходит замена фрагмента на соответствующий функционалу либо вызову диалогового окна. При вызове операций экспорта всех зашифрованных заметок или импорта заметок система проверяет статус аутентификации пользователя. Если

пользователь не авторизован, в контейнер загружается фрагмент авторизации, блокирующий выполнение операции до подтверждения.

Рисунок 18 – Экран настроек

На рисунке 19 представлен фрагмент смены пароля функциональной задачей является обеспечение аутентификации пользователя через проверку учетных данных и предоставлении механизма обновления пароля.

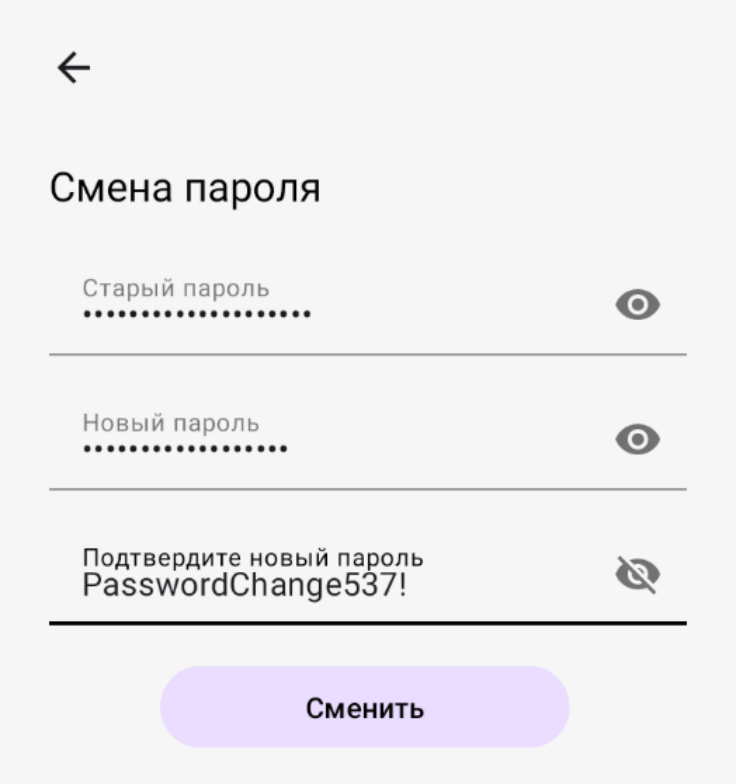

Рисунок 19 – Фрагмент смены пароля

На рисунке 20 представлено диалоговое окно операции экспортировать все зашифрованные заметки из списка меню, предназначенное для получения от пользователя уникального имени файла, в который будут экспортированы все зашифрованные заметки.

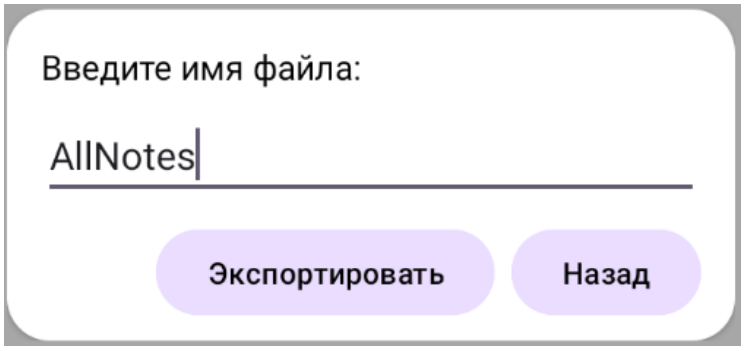

Рисунок 20 – Диалоговое окно операции экспортировать все зашифрованные заметки

На рисунке 21 представлен фрагмент импорта заметок функциональной задачей является обеспечение восстановление конфиденциальной информации из внешнего файла. При активации кнопки «Выбор файла» вызывается системный файловый менеджер, с помощью которого выбирается требуемый файл.

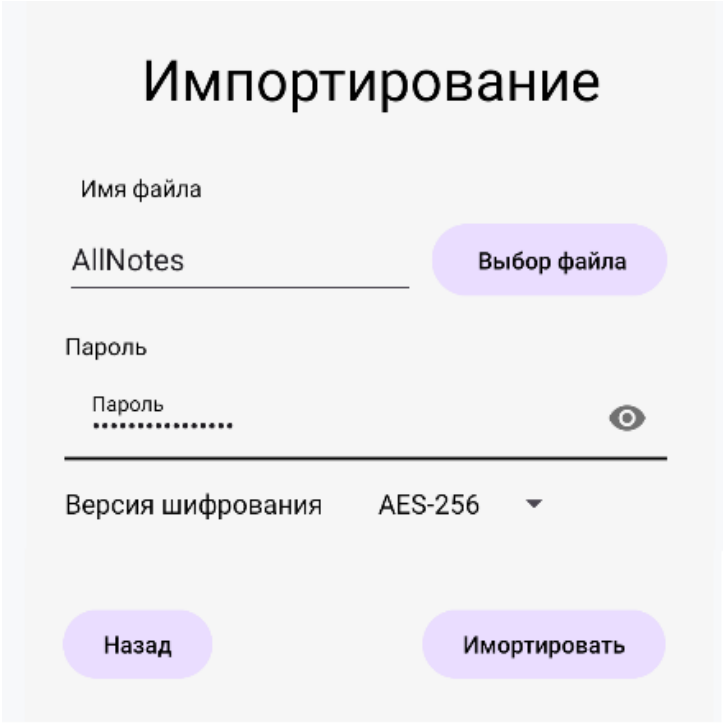

Рисунок 21 – Фрагмент импортирования заметок

4 БЕЗОПАСНОСТЬ И ЭКОЛОГИЧНОСТЬ

4.1 Безопасность

Обеспечение безопасности при разработке и эксплуатации программного обеспечения является одной из ключевых задач, направленной на защиту здоровья пользователей, соблюдение норм охраны труда и предотвращение негативного влияния факторов внешней среды. При создании Android-приложения для создания конфиденциальных заметок необходимо учитывать как физические, так и эргономические аспекты, связанные с использованием вычислительной техники.

Процесс разработки программного обеспечения связан с длительным пребыванием за компьютером, что может привести к переутомлению, зрительному напряжению и нарушению осанки. Конечные пользователи также взаимодействуют с цифровыми устройствами, поэтому важно предусмотреть меры, снижающие риски, связанные с их безопасностью. Нарушение принципов эргономики и правил организации рабочего места может стать причиной снижения работоспособности и общего дискомфорта.

4.1.1 Требования к микроклимату помещения

Для поддержания продуктивности труда и профилактики переутомления критически важно создать стабильные параметры микроклимата в рабочем помещении. Оптимальные температурный режим, уровень влажности и циркуляция воздуха способствуют сохранению физического и психического комфорта, а также снижению риска ухудшения самочувствия.

Для обеспечения безопасных и комфортных условий труда при разработки программного обеспечения параметры микроклимата в помещении должны соответствовать с нормами СанПиН 1.2.3685-21:

– температуру воздуха 22–24 °С в холодный период и 23–25 °С – в теплый;

– относительную влажность воздуха 40–60 %;

– скорость движения воздуха – не более 0,1 м/с.

Для обеспечения указанных условий рекомендуется внедрить комплекс технических решений:

– Использование климатических систем (кондиционеры, обогреватели) для точного контроля температуры;

– Организация эффективной вентиляции для поддержания свежести воздуха;

– Применение увлажнителей или ионизаторов для корректировки влажности в случае отклонения от нормативных значений.

4.1.2 Освещение рабочего места

Освещение рабочего места при работе с персональными электронно-вычислительными машинами (ПЭВМ) имеет первостепенное значение, особенно для специалистов в области разработки программного обеспечения. Постоянное напряжение зрения, вызванное недостаточным или некорректно организованным освещением, может привести к зрительному переутомлению, снижению концентрации, головным болям и даже хроническим нарушениям зрения. В связи с этим соблюдение санитарно-эпидемиологических норм, установленных СанПиН 1.2.3685–21, становится обязательным условием обеспечения здоровых и безопасных условий труда.

Работа с ПЭВМ требует особого подхода к организации освещения, поскольку длительное напряжение глаз приводит к снижению работоспособности и риску развития зрительного утомления. Освещение должно обеспечивать равномерное распределение света без резких перепадов освещенности, исключать блики и отражения на экранах мониторов, а также соответствовать установленным нормам освещенности для компьютерных рабочих мест.

Естественное освещение рабочих зон с ПЭВМ должно быть организовано так, чтобы исключить прямое попадание солнечных лучей на экраны. Мониторы рекомендуется размещать боковой стороной к окнам или под углом 90° к ис-

точнику света. При недостаточной естественной освещенности (коэффициент естественной освещенности – КЕО менее 1,5%) необходимо использовать дополнительные источники искусственного света. Для регулирования интенсивности естественного света применяются жалюзи, шторы или светорассеивающие покрытия на окнах.

Искусственное освещение в помещениях с компьютерами должно обеспечивать общую освещенность не менее 500 лк на уровне 0,8 м от пола (рабочая поверхность) и не менее 300 лк на клавиатуре и документах. Используются люминесцентные или светодиодные лампы с цветовой температурой 4000–4500 К (нейтральный белый свет), которые минимизируют зрительное утомление. Светильники должны быть экранированы матовыми рассеивателями или решетками для предотвращения слепящего эффекта. Коэффициент пульсации светового потока не должен превышать 10% (для помещений с динамической нагрузкой – 5%).

4.1.3 Эргономические требования

Организация рабочего места разработчика должна соответствовать принципам эргономики и физиологическим особенностям человека, чтобы минимизировать риск переутомления, нарушений осанки и заболеваний опорно-двигательного аппарата. СанПиН 1.2.3685–21 устанавливает конкретные нормы для обеспечения оптимальной позы, удобства размещения оборудования и снижения статической нагрузки на тело.

Стол для работы с компьютером должен быть адаптирован к росту и комплекции пользователя. Его стандартная высота составляет 72–76 см , а глубина – не менее 60 см , чтобы обеспечить свободное размещение клавиатуры, мыши, монитора и документов.

Рабочее кресло должна быть регулируемой по высоте, чтобы обеспечить правильное положение тела: стопы полностью стоят на полу или на подставке для ног, колени согнуты под углом 90–110°, а предплечья находятся на уровне клавиатуры. Высота сиденья рекомендуется в диапазоне 40–50 см , а его глуби-

на – 40–45 см , чтобы между краем сиденья и подколенной ямкой оставалось расстояние не менее 5 см. Спинка кресла должна иметь поддержку поясничного отдела позвоночника (поясничный валик), регулируемый наклон и угол опережения, компенсирующий отклонение корпуса назад. Подлокотники должны быть регулируемыми по высоте и ширине, чтобы обеспечивать опору рук без ограничения движений. Для снижения давления на тазобедренные суставы рекомендуется использовать сиденье с закругленным передним краем. В случае длительной работы без возможности регулярной смены позы необходимо предусмотреть наличие подставки для ног , которая позволит разгрузить позвоночник и улучшить кровообращение.

Монитор должен находиться на расстоянии 50–70 см от глаз пользователя, что соответствует длине вытянутой руки. Верхняя граница экрана должна быть на уровне глаз или немного ниже (примерно на 15–20° ниже горизонтального взгляда), чтобы избежать чрезмерного наклона головы. Угол наклона экрана регулируется от 10 до 30° для минимизации бликов и обеспечения прямой линии взгляда.

Клавиатура и мышь размещаются на уровне локтей (высота 60–70 см от пола), под углом 70–110° в локтевых суставах. Запястья во время работы должны оставаться прямыми, без перегибов. Для этого используются съемные подставки или встроенные лотки под клавиатуру.

4.1.4 Организация рабочего времени

Эргономичная организация рабочего времени играет ключевую роль в обеспечении безопасных условий труда при работе с компьютером. Согласно требованиям Министерства труда РФ № 926н, непрерывная работа за монитором не должна превышать 45–60 минут. После этого необходимо делать перерыв продолжительностью от 5 до 15 минут, в течение которого рекомендуется либо полностью отвлечься от экрана, либо выполнить физические упражнения для профилактики переутомления.

Такое чередование нагрузки и отдыха способствует снижению напряжения как в зрительной системе, так и в опорно-двигательном аппарате, а также предотвращает развитие хронической усталости и связанных с ней профессиональных заболеваний.

Одним из наиболее распространённых последствий длительной работы за монитором является зрительное переутомление. Для его профилактики рекомендуется выполнять специальные упражнения, направленные на расслабление глазных мышц и улучшение кровообращения:

- медленные круговые вращения глазами по часовой стрелке и против (по 5 раз в каждую сторону);
- перевод взгляда вправо-влево и вверх-вниз с повторением каждого направления 5–10 раз;
- лёгкое массирование век подушечками пальцев для нормализации оттока жидкости и расслабления окружающих мышц;
- искусственное увеличение частоты моргания в течение 1–2 минут для увлажнения поверхности глаз.

Эти простые действия позволяют значительно снизить риск развития таких состояний, как синдром «сухого глаза», утомляемость сетчатки и нарушение аккомодации.

Долгое нахождение в одной позе часто приводит к напряжению мышц шеи и головы, что может вызывать дискомфорт и даже спровоцировать головную боль. В целях профилактики рекомендуется выполнять следующие упражнения:

- лёгкий массаж висков и затылка подушечками пальцев, способствующий улучшению лимфооттока;
- наклоны головы в стороны с задержкой в крайнем положении на 10–15 секунд – помогает растянуть шейные мышцы;
- подъём и опускание плеч (5–10 повторений) – эффективно снимает напряжение, передаваемое на шею и голову;

– упражнения с сопротивлением: надавливание ладонью на лоб и подбородок с фиксацией положения на 10 секунд – способствует глубокому расслаблению мышц шеи.

При активной работе с клавиатурой и мышью особенно важно разминать кисти и предплечья. Рекомендуемые упражнения:

– вращение кистями по и против часовой стрелки (по 5–7 раз) – улучшает кровообращение и снимает напряжение в запястьях;

– массаж ладони большим пальцем другой руки – способствует расслаблению мышц кисти;

– сгибание и разгибание пальцев (8–10 раз на каждую руку) – укрепляет суставы и снижает усталость;

– разведение пальцев в стороны с задержкой на 10 секунд – выполняется 3–5 раз на каждой руке.

Длительное сидение в одном положении может привести к ослаблению мышц кора, нарушению осанки, болям в спине и даже дыхательным расстройствам. Для компенсации этих эффектов рекомендуется выполнять следующие движения:

– наклоны корпуса вправо и влево с сохранением прямой спины и упором на пятки (по 10 секунд в каждую сторону);

– наклоны вперед с округлённой спиной и свободным опусканием рук (удержание позы – 15–20 секунд);

– пружинящие полуприседания – 5–7 мягких движений вверх-вниз с последующим расслаблением мышц ног и поясницы;

– дыхательная растяжка с вытянутыми руками: на выдохе вытягиваются руки вперёд, ладонями вверх, лопатки сводятся вместе; на вдохе –расслабление и опускание рук. Повторяется 5–7 раз.

Все перечисленные упражнения целесообразно выполнять в рамках так называемых физкультурных пауз – коротких перерывов между рабочими циклами, направленных на восстановление работоспособности. Такие паузы могут

включать как статические, так и динамические упражнения: наклоны головы, вращение плечами, разминку рук и ног, а также ходьбу на месте.

Регулярное выполнение комплекса упражнений позволяет поддерживать мышечный тонус, улучшает осанку, снижает риск возникновения болевых синдромов и повышает общую продуктивность труда.

4.1.5 Эргономика использования мобильных устройств

Правильная эргономика при использовании мобильных устройств играет ключевую роль в предотвращении физического дискомфорта и долгосрочных последствий для здоровья. Основные риски связаны с неправильной осанкой, чрезмерным напряжением мышц шеи и верхних конечностей, а также переутомлением глаз. Для минимизации этих эффектов необходимо учитывать позу тела, положение устройства и продолжительность взаимодействия.

При работе с телефоном сидя за столом рекомендуется держать экран на уровне глаз, чтобы избежать наклона головы вниз, который провоцирует развитие «текст-шеи» – хронического напряжения шейного отдела позвоночника. Локти должны опираться на поверхность стола, а запястья оставаться в нейтральном положении, что снижает риск синдрома запястного канала. Для длительного использования целесообразно применять внешние подставки, которые фиксируют устройство под оптимальным углом.

При сидячем использовании телефона важно не только расположение устройства, но и организация рабочего пространства. Высота стола должна соответствовать длине рук пользователя, чтобы локти образовывали угол 90–100 градусов. Если стол статичен, можно использовать регулируемый стул или подставку для ног, чтобы снизить нагрузку на поясницу.

Использование телефона в вертикальной позе требует особого внимания к равновесию тела и положению позвоночника. Стандартная рекомендация – держать устройство на уровне груди или чуть ниже, что позволяет сохранить прямую осанку и снизить напряжение в шейном и грудном отделах. Однако

длительное фиксирование взгляда на экране во время движения увеличивает риск травм из-за столкновений или падений.

Использование телефона в горизонтальном положении сопряжено с наибольшими рисками для здоровья. Лежа на спине с поднятым устройством под прямым углом к глазам, пользователь вынужден фиксировать взгляд вверх, что вызывает спазмы мышц верхней части лица и сухость роговицы. Кроме того, такое положение ограничивает диафрагмальное дыхание, что может привести к гипоксии тканей.

Лежа на боку, важно избегать вынужденного поворота головы в сторону экрана: оптимально расположить устройство на уровне глаз. Важно также контролировать давление на руку, которая находится под телом: регулярная смена стороны предотвратит онемение и болевые ощущения.

Для снижения зрительного утомления вечером рекомендуется использовать режим «Тёмный экран» (Dark Mode), который может снизить нагрузку на глаза в условиях слабого освещения. Также полезно применять приложения, автоматически изменяющие цветовую температуру экрана в зависимости от времени суток.

Независимо от позы, ключевым элементом эргономики является регулярная смена активности. Всемирная организация здравоохранения рекомендует придерживаться правила 20-20-20: каждые 20 минут отводить взгляд от экрана на 20 секунд, фокусируясь на объекте на расстоянии 6 метров. Это снижает риск синдрома компьютерного зрения, проявляющегося в сухости, покраснении и жжении глаз.

4.1.6 Безопасность использования мобильных устройств

Для обеспечения физической безопасности мобильного телефона и его пользователя требуется соблюдать условия эксплуатации:

– Использование устройства вне температурного диапазона от 0 °C до 40 °C не рекомендуемо, поскольку экстремальные значения температуры могут вызвать необратимые повреждения аккумулятора и других компонентов

– Длительное нахождение телефона под прямыми солнечными лучами не допускается во избежание перегрева и деформации корпуса.

– Процесс зарядки устройства должен осуществляться исключительно с использованием оригинальных зарядных устройств и кабелей, сертифицированных производителем. Продолжительность зарядки не должна превышать необходимого времени, чтобы исключить риск возгорания или короткого замыкания.

– Во время грозы требуется отключать зарядные устройства от электросети.

– Регулярный осмотр зарядных кабелей и адаптеров на наличие повреждений изоляции обязателен, а неисправные аксессуары должны быть заменены.

– Соблюдение указанных требований обеспечивает долговечность устройства, минимизирует риски технических неисправностей и защищает здоровье пользователя.

4.2 Экологичность

С ростом числа мобильных устройств и увеличением нагрузки на их ресурсы вопросы энергоэффективности программного обеспечения приобретают критическое значение. Даже незначительное повышение энергопотребления одного приложения в масштабах миллионов устройств приводит к двум серьезным последствиям:

– Увеличению углеродного следа цифровой инфраструктуры;

– Сокращению времени автономной работы устройств.

Основные источники энергопотребления в мобильных приложениях

– Процессор – особенно при выполнении ресурсоемких операций, таких как шифрование данных;

– Экран – главный потребитель энергии, особенно при высокой яркости.

Выбор алгоритма шифрования: баланс безопасности и эффективности Для приложения, ориентированного на работу с зашифрованным текстом, был

выбран алгоритм AES (Advanced Encryption Standard). Это решение обусловлено сочетанием высокой криптографической стойкости и относительно низкого энергопотребления по сравнению с альтернативными методами.

Технические решения для повышения энергоэффективности

– Оптимизация локального хранения данных Разработана система кэширования расшифрованных заметок в оперативной памяти устройства, минимизирующая обращения к базе данных.

– Изменение кодировки текста Переход с UTF-8 на CP1251 позволил сократить объем хранимых данных вдвое, что уменьшило нагрузку на процессор при шифровании/дешифровании.

Рекомендации пользователям для снижения энергопотребления

- Активируйте режим энергосбережения в настройках устройства, чтобы ограничить фоновые процессы;
- Регулярно обновляйте операционную систему для получения энергоэффективных улучшений;
- Отключайте неиспользуемые беспроводные модули (Wi-Fi, Bluetooth, GPS);
- Снижайте яркость экрана до комфортного уровня.

4.3 Чрезвычайные ситуации

При рассмотрении возможности возникновении чрезвычайных ситуаций нужно отметить то, что они могут возникнуть как при разработки, так и при использовании конечным пользователем приложения.

Чрезвычайные ситуации при разработки программного обеспечения связанны с использованием персональных электронно-вычислительных машин и сопутствующего оборудования, которые при неправильном подключении или неисправности могут стать источником возгорания. Наиболее вероятной ЧС техногенного характера в таких условиях является пожар.

Основные причины возникновения пожара в помещениях с ПЭВМ:

– Неисправность электропроводки или электрооборудования: Короткое замыкание в электросети, розетках, удлинителях; перегрузка электрической сети из-за подключения большого количества мощных потребителей; неисправность блоков питания ПЭВМ или периферийных устройств.

– Нарушение правил эксплуатации электроприборов: Использование поврежденных или кустарных электроприборов, оставление включенных устройств без присмотра на длительное время, перекрытие вентиляционных отверстий оборудования, что приводит к перегреву.

– Возгорание легковоспламеняющихся материалов: Хранение бумаги, упаковочных материалов, горючих жидкостей в непосредственной близости от нагревательных приборов или источников искрения.

Меры по предупреждению пожара в помещениях, где ведется разработка:

Обеспечение исправности электросети и электрооборудования: Регулярная проверка состояния электропроводки, розеток, выключателей и удлинителей. Использование только сертифицированного и технически исправного электрооборудования. Запрет на использование самодельных или поврежденных электроприборов. Внедрение автоматических выключателей для предотвращения перегрузок сети.

Контроль за эксплуатацией оборудования: Обеспечение регулярного технического обслуживания ПЭВМ и периферийных устройств. Запрет на длительное бесконтрольное использование техники, установка программных таймеров для автоматического отключения устройств. Организация доступа к вентиляционным отверстиям оборудования для предотвращения перегрева.

Организация хранения легковоспламеняющихся материалов: Запрет на хранение легковоспламеняющихся веществ (ЛВМ) в непосредственной близости от электроники или источников тепла.

Использование источников бесперебойного питания (ИБП): Подключение ПЭВМ и критически важного оборудования к сертифицированным ИБП для защиты от скачков напряжения, перегрузок и внезапных отключений электро-

энергии. Регулярное техническое обслуживание ИБП, включая замену аккумуляторов, для предотвращения перегрева или возгорания самих устройств.

Мобильные устройства включают компоненты, которые при неисправности или неправильной эксплуатации могут стать причиной пожара. К ним относятся:

– Литий-ионные аккумуляторы: Наиболее уязвимый элемент, склонный к перегреву, вздутию и термическому разгону при повреждении, перезарядке или использовании некачественных зарядных устройств.

– Зарядные устройства и кабели: Неисправные или несертифицированные аксессуары могут вызвать короткое замыкание, перегрузку цепи или искрение.

Меры по предупреждению пожара при использовании мобильных устройств:

− Использование сертифицированных аксессуаров: Подключение к зарядным устройствам и кабелям, соответствующим стандартам безопасности. Не рекомендуется применение самодельных, поврежденных или неоригинальных аксессуаров.

− Правила безопасной зарядки: Запрет на размещение устройств на текстильных изделиях (подушки, покрывала), синтетических покрытиях при зарядке мобильного устройства. Это обусловлено тем, что данные материалы способны аккумулировать тепло, усиливая термическое воздействие на корпус устройства и повышая вероятность воспламенения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы было разработано мобильное приложение для создания и безопасного хранения текстовых заметок. Основной задачей мобильного приложения являлось минимизация рисков несанкционированного доступа к данным счет реализации локального шифрования на устройстве пользователя.

В процессе выполнения бакалаврской работы были выполнены следующие задачи: проведён анализ криптографических стандартов, обоснован выбор инструментов разработки, спроектирована архитектура приложения, реализованы механизмы шифрования и дешифрования данных, а такте создан интуитивно понятный интерфейс пользователя для мобильного приложения. Таким образом, все этапы, запланированные на начальном этапе исследования, выполнены в полном объёме.

Основой защиты данных стало применение стандарта AES с различными длинами ключей (128, 192, 256 бит), что позволило высокую степень конфиденциальности пользовательских заметок. Реализация алгоритма AES выполнена строго в соответствии с анализом.

Разработанное приложение реализует функционал для создания, редактирования и удаления текстовых заметок через интуитивно понятный интерфейс, обеспечивающий удобство взаимодействия пользователя с системой, при этом данные шифруются по алгоритму AES и сохраняются локально на устройстве.

Несмотря на успешную — реализацию базовых функций, программ обладает потенциалом для дальнейшего совершенствования. В дальнейшем планируется повышение криптостойкость зашифрованных заметок за счёт внедрения режимов блочного шифрования, и улучшение пользовательского функционала в редактировании текста за счёт добавления выбора различных шрифтов, выделения и размера текста.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аграновский, А. В. Практическая криптография: алгоритмы и их программирование / А. В. Аграновский, Р. А. Хади. — Москва : СОЛОН-Пресс, 2021. — 256 с.

2. Аделекан И. Kotlin: программирование на примерах: Пер. с англ. / И. Аделекан. - Санкт-Петербург : БХВ-Петербург, 2020. - 432 с.

3. Баланов А. Н. Комплексное руководство по разработке: от мобильных приложений до веб-технологий : учебное пособие для вузов / А. Н. Баланов. — 2-е изд., стер. — Санкт-Петербург : Лань, 2025. — 412 с.

4. Баланов, А. Н. Цифровые платформы и системы : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — 452 с.

5. Басалова Г. В. Основы криптографии : учебное пособие / Басалова Г. В. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 282 с

6. Борисенков, Д. В. Расширенные возможности языка SQL : учебное пособие / Д. В. Борисенков, Ю. А. Крыжановская. — Воронеж : ВГУ, 2021. — 31с

7. Васильева, И. Н. Криптографические методы защиты информации : учебник и практикум для вузов / И. Н. Васильева. — Москва : Издательство Юрайт, 2025. — 310 с

8. Золкин, А. Л. Кросс-платформенное программирование в системах реального времени : учебник для вузов / А. Л. Золкин. — Санкт-Петербург : Лань, 2025. — 152 с.

9. Игнатьев, Е. Б. Симметричные блочные криптоалгоритмы: учеб. пособие / Е. Б. Игнатьев; ФГБОУ ВО «Иван. гос. энергет. ун-т им. В. И. Ленина».– Иваново, 2021. – 176 с

10. Иванюгин, В. М. Основы информационной безопасности. Алгоритмы шифрования данных : методические указания / В. М. Иванюгин. — Москва : РТУ МИРЭА, 2021. — 30 с.

11. Калгина, И. С. Разработка мобильных приложений : учебное пособие / И. С. Калгина. — Чита : ЗабГУ, 2022. — 163 с.

12. Казарин, О. В. Программно-аппаратные средства защиты информации. Защита программного обеспечения : учебник и практикум для вузов / О. В. Казарин, А. С. Забабурин. — Москва : Издательство Юрайт, 2025. — 312 с.

13. Киреев, Н. В. Разработка мобильных приложений на Котлин : учебное пособие / Н. В. Киреев. — Красноярск : СибГУ им. академика М. Ф. Решетнёва, 2023. — 80 с.

14. Коузен, К. Kotlin. Сборник рецептов / К. Коузен ; перевод с английского А. Н. Киселева. — Москва : ДМК Пресс, 2021. — 220 с.

15. Краковский, Ю. М. Методы защиты информации : учебное пособие для вузов / Ю. М. Краковский. — 3-е изд., перераб. — Санкт-Петербург : Лань, 2021. — 236 с.

16. Лось, А. Б. Криптографические методы защиты информации для изучающих компьютерную безопасность : учебник для вузов / А. Б. Лось, А. Ю. Нестеренко, М. И. Рожков. — 2-е изд., испр. — Москва : Издательство Юрайт, 2025. — 424 ср

17. Льюис, Ш. Нативная разработка мобильных приложений / Ш. Льюис, М. Данн ; перевод А. Н. Киселев. — Москва : ДМК Пресс, 2020. — 376 с

18. Нестеров, С. А. Базы данных : учебник и практикум для вузов / С. А. Нестеров. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 258 с.

19. Петросян, Л. Э. Разработка мобильных приложений на языке Kotlin : учебное пособие для вузов / Л. Э. Петросян, К. В. Гусев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2025. — 104 с.

20. Пирская, Л. В. Разработка мобильных приложений в среде Android Studio : учебное пособие / Л. В. Пирская. — Ростов-на-Дону : ЮФУ, 2019. — 123 с.

21. Поляков, Е. Ю. Введение в векторную графику / Е. Ю. Поляков. — 2-е изд., стер. (полноцветная печать). — Санкт-Петербург : Лань, 2023. — 256 с.

22. Приказ Минтруда России от 24.12.2021 № 926н «Об утверждении правил по охране труда при работе с персональными электронно-вычислительными машинами»: зарегистрирован в Минюсте РФ 29.12.2021 № 66883. М., Минтруд России, 2021. 36 с.

23. Рацеев, С. М. Криптографические методы защиты информации и их основы. Лабораторный практикум : учебное пособие для вузов / С. М. Рацеев. — Санкт-Петербург : Лань, 2025. — 148 с.

24. СанПиН 1.2.3685-21. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания : утв. постановлением Главного государственного санитарного врача РФ от 28 января 2021 г. № 2 : введен в действие с 01 марта 2021 г.

25. Синицын, И. В. Разработка мобильных приложений : учебное пособие / И. В. Синицын, Е. А. Чернов, Ю. А. Воронцов. — Москва : РТУ МИРЭА, 2023. — 162 с

26. Соколова, В. В. Разработка мобильных приложений : учебник для среднего профессионального образования / В. В. Соколова. — Москва : Издательство Юрайт, 2025. — 160 с.

27. Сомон., П. И. Волшебство Kotlin : руководство / П. И. Сомон. ; перевод с английского А. Н. Киселева.. — Москва : ДМК Пресс, 2020. — 536 с.

28. СП 52.13330.2016. Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95* : утв. приказом Министерства строительства и жилищно-коммунального хозяйства РФ от 07 ноября 2016 г. №777/пр : введен в действие с 08 мая 2017 г.