**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**Лабораторная работа № 7**

**Дисциплина: Тестирование программного обеспечения**

**Тема: Тестирование мобильного приложения**

**Выполнил: студент группы 221-371**

**Евдокимов Артемий Русланович**

**(Фамилия И.О )**

**Дата, подпись \_\_22.09.2025\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**(Фамилия И.О., степень, звание)**

**Дата, подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_***

**(Дата) (Подпись)**

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2025**

**1. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ**

**1.1. Основной функционал**

* **Запуск приложения**
  + Приложение запускается без ошибок
  + Отображается главный экран с полями ввода
  + Кнопки активны и кликабельны
* **Создание заметки**
  + Ввод заголовка и текста
  + Сохранение заметки кнопкой "Сохранить заметку"
  + Проверка отображения в списке
  + Очистка полей после сохранения
* **Валидация полей ввода**
  + Попытка сохранения с пустым заголовком
  + Попытка сохранения с пустым текстом
  + Кнопка "Сохранить" неактивна при пустых полях

**1.2. Управление заметками**

* **Просмотр заметок**
  + Отображение списка сохраненных заметок
  + Корректное отображение заголовка и текста
  + Отображение времени создания
* **Удаление заметок**
  + Удаление через кнопку удаления (иконка корзины)
  + Подтверждение удаления через диалог
  + Обновление списка после удаления
* **Информация о заметке**
  + Долгое нажатие для показа информации
  + Отображение времени создания в Toast

**2. ТЕСТИРОВАНИЕ ЖЕСТОВ**

**2.1. Базовые жесты**

* **Тап**
  + Тап по кнопке сохранения
  + Тап по кнопкам удаления/информации
  + Тап по элементам списка
* **Долгое нажатие** 
  + Долгое нажатие на карточку заметки → информация
* **Свайп** 
  + Вертикальный свайп для прокрутки списка
  + Плавность анимации прокрутки

**3. ТЕСТИРОВАНИЕ ОРИЕНТАЦИИ ЭКРАНА**

**3.1. Смена ориентации**

* **Портрет → Ландшафт**
  + Поворот устройства/эмулятора
  + Корректное отображение интерфейса
* **Ландшафт → Портрет**
  + Обратный поворот
  + Корректное позиционирование элементов

**4. ТЕСТИРОВАНИЕ ПРЕРЫВАНИЙ**

**4.1. Входящие вызовы**

* **Звонок во время работы приложения (adb shell am start -a android.intent.action.CALL -d tel:5551234)**
  + Имитация входящего звонка через Extended Controls
  + Поведение приложения во время звонка
  + Восстановление после завершения звонка

**4.2. SMS/Уведомления**

* **Входящее SMS(adb emu sms send 5551234 "Тестовое сообщение для лабораторной работы")**
  + Имитация SMS через adb
  + Поведение уведомления
  + Восстановление после просмотра SMS

**4.3. Системные прерывания**

* **Переход в фоновый режим**
  + Нажатие кнопки Home
  + Возврат в приложение
  + Сохранение состояния

**Листинг кода всех приложений и тестов:**

**test.py:**

import os

import time

import subprocess

from appium import webdriver

from appium.options.common import AppiumOptions

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

class SimpleNotesTests:

    def \_\_init\_\_(self):

        os.environ['ANDROID\_HOME'] = "D:\\First\_Android\_App"

        os.environ['ANDROID\_SDK\_ROOT'] = "D:\\First\_Android\_App"

        self.driver = None

        self.wait = None

    def setup(self):

        print("Настройка тестового окружения...")

        options = AppiumOptions()

        options.load\_capabilities({

            'platformName': 'Android',

            'appium:deviceName': 'emulator-5554',

            'appium:automationName': 'UiAutomator2',

            'appium:appPackage': 'com.example.simplenotes',

            'appium:appActivity': '.MainActivity',

            'appium:noReset': False

        })

        try:

            self.driver = webdriver.Remote(

                command\_executor='http://127.0.0.1:4723/wd/hub',

                options=options

            )

            self.wait = WebDriverWait(self.driver, 10)

            time.sleep(5)

            print("✓ Драйвер успешно настроен")

            return True

        except Exception as e:

            print(f"✗ Ошибка настройки драйвера: {e}")

            return False

def teardown(self):

        if self.driver:

            self.driver.quit()

            print("✓ Драйвер закрыт")

def find\_by\_text(self, text):

        """Поиск элемента по тексту"""

        try:

            return self.driver.find\_element(By.XPATH, f"//\*[contains(@text, '{text}')]")

        except:

            return None

def create\_test\_note(self):

        """Вспомогательный метод для создания тестовой заметки"""

        try:

            # Находим поля ввода

            input\_fields = self.driver.find\_elements(By.CLASS\_NAME, "android.widget.EditText")

            if len(input\_fields) >= 2:

                # Вводим заголовок

                title\_field = input\_fields[0]

                title\_field.send\_keys("Тестовая заметка")

               # Вводим текст заметки

                content\_field = input\_fields[1]

                content\_field.send\_keys("Это тестовое содержание заметки")

                # Сохраняем заметку

                save\_button = self.find\_by\_text("Сохранить заметку")

                if save\_button and save\_button.is\_enabled():

                    save\_button.click()

                    time.sleep(2)

                    return True

            return False

        except Exception as e:

            print(f"Ошибка создания заметки: {e}")

            return False

    def test\_app\_launch(self):

        """1.1.1 Тест запуска приложения"""

        print("\n1. Тест запуска приложения")

        try:

            # Проверяем основные элементы интерфейса

            title = self.find\_by\_text("Заметки")

            if title:

                print("✓ Заголовок приложения отображается")

            # Проверяем поля ввода

            input\_fields = self.driver.find\_elements(By.CLASS\_NAME, "android.widget.EditText")

            print(f"✓ Найдено полей ввода: {len(input\_fields)}")

            # Проверяем кнопку сохранения

            save\_button = self.find\_by\_text("Сохранить заметку")

            if save\_button:

                print("✓ Кнопка сохранения найдена")

            self.driver.save\_screenshot("test\_launch.png")

            return True

        except Exception as e:

            print(f"✗ Ошибка теста запуска: {e}")

            return False

    def test\_create\_note(self):

        """1.1.2 Тест создания заметки"""

        print("\n2. Тест создания заметки")

        try:

            # Находим поля ввода

            input\_fields = self.driver.find\_elements(By.CLASS\_NAME, "android.widget.EditText")

            if len(input\_fields) >= 2:

                # Вводим заголовок

                title\_field = input\_fields[0]

                title\_field.send\_keys("Тестовая заметка")

                print("✓ Заголовок введен")

                # Вводим текст заметки

                content\_field = input\_fields[1]

                content\_field.send\_keys("Это тестовое содержание заметки")

                print("✓ Текст заметки введен")

# Вводим текст заметки

content\_field = input\_fields[1]

content\_field.send\_keys("Это тестовое содержание заметки")

print("✓ Текст заметки введен")

# Сохраняем заметку

save\_button = self.find\_by\_text("Сохранить заметку")

if save\_button and save\_button.is\_enabled():

save\_button.click()

print("✓ Кнопка сохранения нажата")

# Ждем сохранения

time.sleep(2)

# Проверяем, что поля очистились

if title\_field.text == "" and content\_field.text == "":

print("✓ Поля очистились после сохранения")

# Проверяем, что заметка появилась в списке

note\_title = self.find\_by\_text("Тестовая заметка")

if note\_title:

print("✓ Заметка отображается в списке")

self.driver.save\_screenshot("test\_create\_note.png")

return True

else:

print("✗ Заметка не найдена в списке")

return False

else:

print("✗ Кнопка сохранения не активна")

return False

else:

print("✗ Не найдены поля ввода")

return False

except Exception as e:

print(f"✗ Ошибка создания заметки: {e}")

return False

def test\_validation(self):

"""1.1.3 Тест валидации полей"""

print("\n3. Тест валидации полей")

try:

save\_button = self.find\_by\_text("Сохранить заметку")

if save\_button:

# Проверяем состояние кнопки при пустых полях

is\_enabled = save\_button.is\_enabled()

if not is\_enabled:

print("✓ Кнопка неактивна при пустых полях")

else:

print("✗ Кнопка активна при пустых полях")

# Заполняем только заголовок

                input\_fields = self.driver.find\_elements(By.CLASS\_NAME, "android.widget.EditText")

                if input\_fields:

                    input\_fields[0].send\_keys("Только заголовок")

                    # Даем время на обновление состояния

                    time.sleep(1)

                    # Проверяем состояние кнопки после ввода заголовка

                    is\_enabled\_after\_title = save\_button.is\_enabled()

                    if not is\_enabled\_after\_title:

                        print("✓ Кнопка неактивна без содержания")

                    else:

                        print("✗ Кнопка активна без содержания")

                    # Очищаем поле

                    input\_fields[0].clear()

                    self.driver.save\_screenshot("test\_validation.png")

                    return True

                else:

                    print("✗ Не найдены поля ввода")

                    return False

            else:

                print("✗ Не найдена кнопка сохранения")

                return False

        except Exception as e:

            print(f"✗ Ошибка теста валидации: {e}")

            return False

    def test\_note\_management(self):

        """1.2 Тест управления заметками"""

        print("\n4. Тест управления заметками")

        try:

            # Создаем заметку для тестирования (без вызова полноценного теста)

            if not self.create\_test\_note():

                print("✗ Не удалось создать заметку для тестирования")

                return False

            time.sleep(2)

            # Ищем кнопку информации (по content-desc или по описанию)

            info\_buttons = self.driver.find\_elements(By.XPATH, "//\*[@content-desc='Информация' or @content-desc='Info']")

            if not info\_buttons:

                # Пробуем найти по классу или другим атрибутам

                info\_buttons = self.driver.find\_elements(By.XPATH, "//android.widget.Button[contains(@content-desc, 'информа')]")

            if info\_buttons:

                info\_buttons[0].click()

                print("✓ Кнопка информации нажата")

                time.sleep(1)

# Ищем кнопку удаления

            delete\_buttons = self.driver.find\_elements(By.XPATH, "//\*[@content-desc='Удалить' or @content-desc='Delete']")

            if not delete\_buttons:

                delete\_buttons = self.driver.find\_elements(By.XPATH, "//android.widget.Button[contains(@content-desc, 'удал')]")

            if delete\_buttons:

                delete\_buttons[0].click()

                print("✓ Кнопка удаления нажата")

                # Ждем появления диалога подтверждения

                time.sleep(2)

                # Подтверждаем удаление

                confirm\_button = self.find\_by\_text("Удалить")

                if confirm\_button:

                    confirm\_button.click()

                    print("✓ Подтверждение удаления выполнено")

                    # Ждем удаления и проверяем

                    time.sleep(3)

                    # Проверяем, что заметка удалилась

                    note\_after\_delete = self.find\_by\_text("Тестовая заметка")

                    if not note\_after\_delete:

                        print("✓ Заметка успешно удалена")

                    else:

                        # Дополнительная проверка - может быть другая заметка с таким же текстом

                        print("⚠ Заметка все еще отображается, проверяем количество заметок...")

                        # Можно добавить дополнительную логику проверки

                    self.driver.save\_screenshot("test\_management.png")

                    return True

                else:

                    print("✗ Не найдена кнопка подтверждения удаления")

                    return False

            else:

                print("✗ Не найдены кнопки действий")

                return False

        except Exception as e:

            print(f"✗ Ошибка теста управления: {e}")

            return False

def test\_gestures(self):

        """2.1 Тест жестов - упрощенная версия без TouchAction"""

        print("\n5. Тест жестов")

        try:

            # Создаем заметку для тестирования жестов (без вызова полноценного теста)

            if not self.create\_test\_note():

                print("✗ Не удалось создать заметку для тестирования жестов")

                return False

            time.sleep(2)

# Вместо долгого нажатия тестируем обычное нажатие на заметку

            note\_element = self.find\_by\_text("Тестовая заметка")

            if note\_element:

                note\_element.click()

                print("✓ Нажатие на заметку выполнено")

                time.sleep(1)

            # Тестируем прокрутку - создаем еще несколько заметок

            for i in range(2):

                input\_fields = self.driver.find\_elements(By.CLASS\_NAME, "android.widget.EditText")

                if len(input\_fields) >= 2:

                    input\_fields[0].send\_keys(f"Заметка {i+1}")

                    input\_fields[1].send\_keys(f"Содержание {i+1}")

                    save\_button = self.find\_by\_text("Сохранить заметку")

                    if save\_button and save\_button.is\_enabled():

                        save\_button.click()

                        time.sleep(1)

            # Прокручиваем список с помощью swipe

            window\_size = self.driver.get\_window\_size()

            start\_x = window\_size['width'] // 2

            start\_y = window\_size['height'] \* 3 // 4

            end\_y = window\_size['height'] // 4

            self.driver.swipe(start\_x, start\_y, start\_x, end\_y, 1000)

            print("✓ Прокрутка выполнена")

            time.sleep(2)

            # Проверяем, что можем найти созданные заметки после прокрутки

            for i in range(1, 3):

                note = self.find\_by\_text(f"Заметка {i}")

                if note:

                    print(f"✓ Заметка {i} найдена после прокрутки")

            self.driver.save\_screenshot("test\_gestures.png")

            return True

        except Exception as e:

            print(f"✗ Ошибка теста жестов: {e}")

            return False

def test\_interruptions(self):

        """4. Тест прерываний"""

        print("\n6. Тест прерываний")

        try:

            # Имитируем входящий звонок через ADB

            print("Имитация входящего звонка...")

            result = subprocess.run([

                "adb", "shell", "am", "start", "-a", "android.intent.action.CALL", "-d", "tel:5551234"

            ], capture\_output=True, text=True, timeout=10)

            print(f"Результат звонка: {result.returncode}")

            time.sleep(3)

# Возвращаемся в приложение

self.driver.back()

print("✓ Возврат в приложение после звонка")

time.sleep(2)

# Имитируем низкий заряд батареи

print("Имитация низкого заряда батареи...")

result = subprocess.run([

"adb", "shell", "dumpsys", "battery", "set", "level", "15"

], capture\_output=True, text=True, timeout=10)

print(f"Результат изменения батареи: {result.returncode}")

time.sleep(2)

# Проверяем, что приложение все еще работает

current\_activity = self.driver.current\_activity

if "MainActivity" in current\_activity:

print("✓ Приложение работает после прерываний")

self.driver.save\_screenshot("test\_interruptions.png")

return True

except Exception as e:

print(f"✗ Ошибка теста прерываний: {e}")

return False

def run\_all\_tests(self):

"""Запуск всех тестов"""

print("ЗАПУСК АВТОТЕСТОВ ДЛЯ ПРИЛОЖЕНИЯ ЗАМЕТКИ")

print("=" \* 50)

if not self.setup():

print("Не удалось настроить тестовое окружение")

return

test\_results = []

# Запускаем тесты по порядку

        tests = [

            ("Запуск приложения", self.test\_app\_launch),

            ("Создание заметки", self.test\_create\_note),

            ("Валидация полей", self.test\_validation),

            ("Управление заметками", self.test\_note\_management),

            ("Тест жестов", self.test\_gestures),

            ("Тест прерываний", self.test\_interruptions)

        ]

        for test\_name, test\_method in tests:

            try:

                print(f"\n--- Выполнение теста: {test\_name} ---")

                result = test\_method()

                test\_results.append((test\_name, result))

                time.sleep(2)

            except Exception as e:

                print(f"Ошибка выполнения теста {test\_name}: {e}")

                test\_results.append((test\_name, False))

# Выводим результаты

        print("\n" + "=" \* 50)

        print("РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:")

        print("=" \* 50)

        passed = 0

        for test\_name, result in test\_results:

            status = "✓ ПРОЙДЕН" if result else "✗ НЕ ПРОЙДЕН"

            print(f"{test\_name}: {status}")

            if result:

                passed += 1

        print(f"\nИтого: {passed}/{len(tests)} тестов пройдено")

        # Завершаем

        self.teardown()

# Запуск тестов

if \_\_name\_\_ == "\_\_main\_\_":

    tester = SimpleNotesTests()

    tester.run\_all\_tests()

**MainActivity.kt:**

package com.example.simplenotes  
  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.foundation.rememberScrollState  
import androidx.compose.foundation.verticalScroll  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Delete*import androidx.compose.material.icons.filled.*Info*import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.tooling.preview.Preview  
import androidx.compose.ui.unit.dp  
import android.widget.Toast  
import com.example.simplenotes.ui.theme.SimpleNotesTheme  
import java.io.IOException

class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** SimpleNotesTheme **{** Surface(  
 modifier = Modifier.*fillMaxSize*(),

color = MaterialTheme.colorScheme.background  
 ) **{** NotesApp()  
 **}  
 }  
 }** }  
}  
  
data class Note(  
 val id: Int,  
 val title: String,  
 val content: String,  
 val timestamp: Long = System.currentTimeMillis()  
)  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun NotesApp() {  
 var title by remember **{** *mutableStateOf*("") **}** var content by remember **{** *mutableStateOf*("") **}** val notes = remember **{** *mutableStateListOf*<Note>() **}** var nextId by remember **{** *mutableStateOf*(1) **}** var showDeleteDialog by remember **{** *mutableStateOf*(false) **}** var noteToDelete by remember **{** *mutableStateOf*<Note?>(null) **}** val context = *LocalContext*.current  
 val scrollState = rememberScrollState()  
  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*verticalScroll*(scrollState)  
 .*padding*(16.*dp*),  
 verticalArrangement = Arrangement.spacedBy(16.*dp*)  
 ) **{** // Заголовок приложения  
 Text(  
 text = "Заметки",  
 style = MaterialTheme.typography.headlineSmall  
 )  
  
 // Разделитель  
 Divider()

// Поле для заголовка заметки  
OutlinedTextField(  
 value = title,  
 onValueChange = **{** title = **it }**,  
 label = **{** Text("Заголовок заметки") **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 singleLine = true  
)

// Поле для содержания заметки  
OutlinedTextField(  
 value = content,  
 onValueChange = **{** content = **it }**,  
 label = **{** Text("Текст заметки") **}**,

label = **{** Text("Текст заметки") **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(120.*dp*),  
 maxLines = 4  
)  
  
// Кнопка сохранения  
Button(  
 onClick = **{** if (title.*isNotBlank*() && content.*isNotBlank*()) {  
 notes.add(Note(nextId, title, content))  
 nextId++  
 title = ""  
 content = ""  
 Toast.makeText(context, "Заметка сохранена!", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}**,  
 modifier = Modifier.*align*(Alignment.End),  
 enabled = title.*isNotBlank*() && content.*isNotBlank*()  
) **{** Text("Сохранить заметку")  
**}**// Разделитель  
Divider()  
  
// Список заметок  
Text(  
 text = "Сохраненные заметки (${notes.size}):",  
 style = MaterialTheme.typography.titleMedium  
)  
  
if (notes.isEmpty()) {  
 Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** Text(  
 text = "Заметок пока нет",  
 style = MaterialTheme.typography.bodyMedium,  
 color = MaterialTheme.colorScheme.onSurfaceVariant  
 )  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
 Text(  
 text = "Создайте первую заметку для тестирования",  
 style = MaterialTheme.typography.bodySmall  
 )  
 **}**

} else {  
 LazyColumn(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*heightIn*(max = 400.*dp*),

verticalArrangement = Arrangement.spacedBy(8.*dp*)  
 ) **{** *items*(  
 items = notes,  
 key = **{ it**.id **}** ) **{** note **->** NoteCard(  
 note = note,  
 onDelete = **{** noteToDelete = note  
 showDeleteDialog = true  
 **}**,  
 onInfo = **{** Toast.makeText(  
 context,  
 "Заметка создана: ${android.text.format.DateFormat.format("dd.MM.yyyy HH:mm", note.timestamp)}",  
 Toast.*LENGTH\_LONG* ).show()  
 **}** )  
 **}  
 }** }  
  
 Spacer(modifier = Modifier.*height*(32.*dp*))  
**}**// Диалог подтверждения удаления  
if (showDeleteDialog) {  
 AlertDialog(  
 onDismissRequest = **{** showDeleteDialog = false **}**,  
 title = **{** Text("Удаление заметки") **}**,  
 text = **{** Text("Вы уверены, что хотите удалить заметку \"${noteToDelete?.title}\"?") **}**,  
 confirmButton = **{**

TextButton(  
 onClick = **{** noteToDelete?.*let* **{** notes.remove(**it**) **}** showDeleteDialog = false  
 Toast.makeText(context, "Заметка удалена", Toast.*LENGTH\_SHORT*).show()  
 **}** ) **{** Text("Удалить")  
 **}  
 }**,  
 dismissButton = **{** TextButton(  
 onClick = **{** showDeleteDialog = false **}** ) **{** Text("Отмена")  
 **}  
 }** )  
 }  
}

@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun NoteCard(  
 note: Note,  
 onDelete: () -> Unit,  
 onInfo: () -> Unit  
) {  
 Card(  
 modifier = Modifier  
 .*fillMaxWidth*(),  
 elevation = CardDefaults.cardElevation(defaultElevation = 2.*dp*)  
 ) **{** Row(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*),  
 verticalAlignment = Alignment.CenterVertically,  
 horizontalArrangement = Arrangement.SpaceBetween  
 ) **{** Column(  
 modifier = Modifier.*weight*(1f)  
 ) **{** Text(  
 text = note.title,  
 style = MaterialTheme.typography.titleMedium,  
 color = MaterialTheme.colorScheme.primary  
 )  
 Spacer(modifier = Modifier.*height*(4.*dp*))  
 Text(  
 text = note.content,  
 style = MaterialTheme.typography.bodyMedium  
 )  
 Spacer(modifier = Modifier.*height*(4.*dp*))  
 Text(  
 text = android.text.format.DateFormat.format("dd.MM.yyyy HH:mm", note.timestamp).toString(),  
 style = MaterialTheme.typography.labelSmall,  
 color = MaterialTheme.colorScheme.onSurfaceVariant  
 )  
 **}**

// Кнопки действий  
Row **{** IconButton(  
 onClick = onInfo,  
 modifier = Modifier.*size*(24.*dp*)  
 ) **{** Icon(  
 imageVector = Icons.Default.*Info*,  
 contentDescription = "Информация",  
 tint = MaterialTheme.colorScheme.primary  
 )  
 **}**

Spacer(modifier = Modifier.*width*(8.*dp*))  
IconButton(

onClick = onDelete,  
 modifier = Modifier.*size*(24.*dp*)

# Выводим результаты

        print("\n" + "=" \* 50)

        print("РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:")

        print("=" \* 50)

        passed = 0

        for test\_name, result in test\_results:

            status = "✓ ПРОЙДЕН" if result else "✗ НЕ ПРОЙДЕН"

            print(f"{test\_name}: {status}")

            if result:

                passed += 1

        print(f"\nИтого: {passed}/{len(tests)} тестов пройдено")

        # Завершаем

        self.teardown()

# Запуск тестов

if \_\_name\_\_ == "\_\_main\_\_":

    tester = SimpleNotesTests()

    tester.run\_all\_tests()

**AndroidManifest.xml**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <!-- Разрешение для выполнения команд -->  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.SimpleNotes"  
 tools:targetApi="31">  
  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.SimpleNotes">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
</manifest>