Отчет по практической работе

|  |  |
| --- | --- |
| Дисциплина | МДК |
| Группа | 413 |
| ФИО | Киселева Алина Витальевна |
| Дата | 30.10 |

## 1 Интерфейс пользователя

Приложение состоит из 4 основных Activity:

Authorization (авторизация) – стартовая Activity, которая служит для ввода данных учетной записи пользователя;

Registation (регистрация) – Activity регистрации, которая служит для ввода данных при создании новой учетной записи;

Notes (заметки) – Activity меню заметок, на нем показан список всех заметок пользователя, а также ряд возможных активностей с ними (удаление, изменение, создание);

Note (заметка) – Activity заметки, служит для создания или изменения заметки.

На рисунке 1 показана схема переходов между Activity при нажатии соответствующих кнопок.

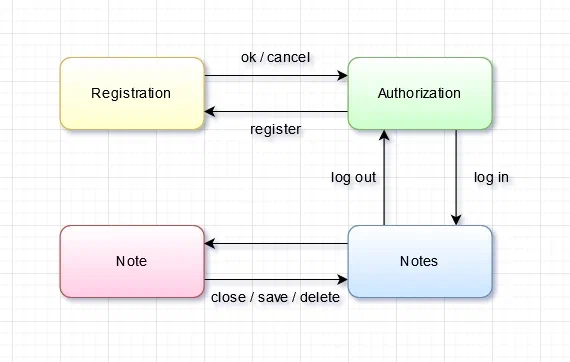


Рисунок 1 -- Схема переходов между Activity

## 1.1. Авторизация пользователя

На рисунке 2 показан внешний вид формы авторизации пользователя.

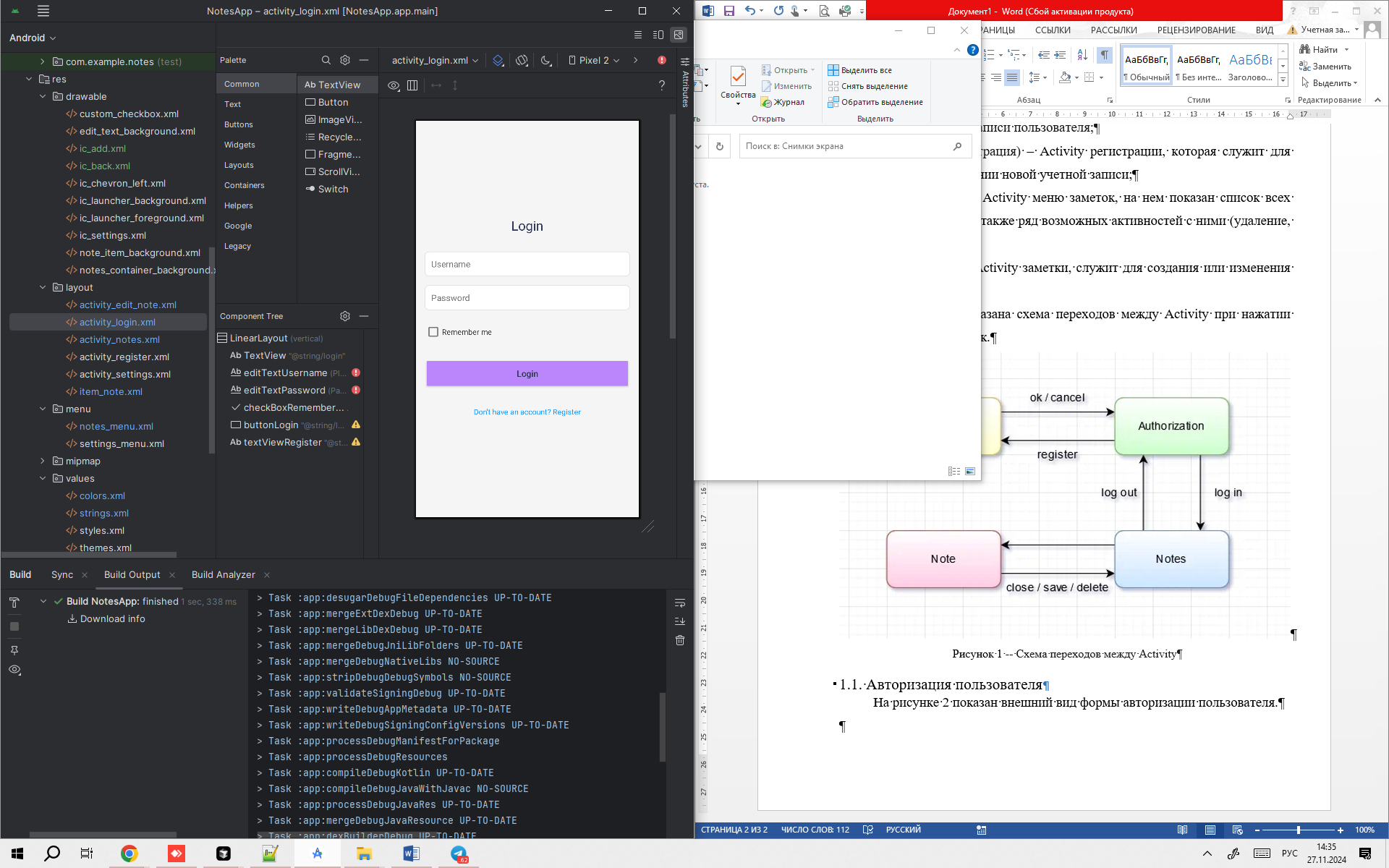


Рисунок 2 -- Activity для авторизации пользователя

На экране присутствуют компоненты EditText для ввода имени пользователя и пароля,

компонент CheckBox для возможности сохранения данных учетной записи в локальной базе данных MySQL и кнопки входа и регистрации.

1.2. Регистрация нового пользователя

На рисунке 3 показан внешний вид формы регистрации пользователя.

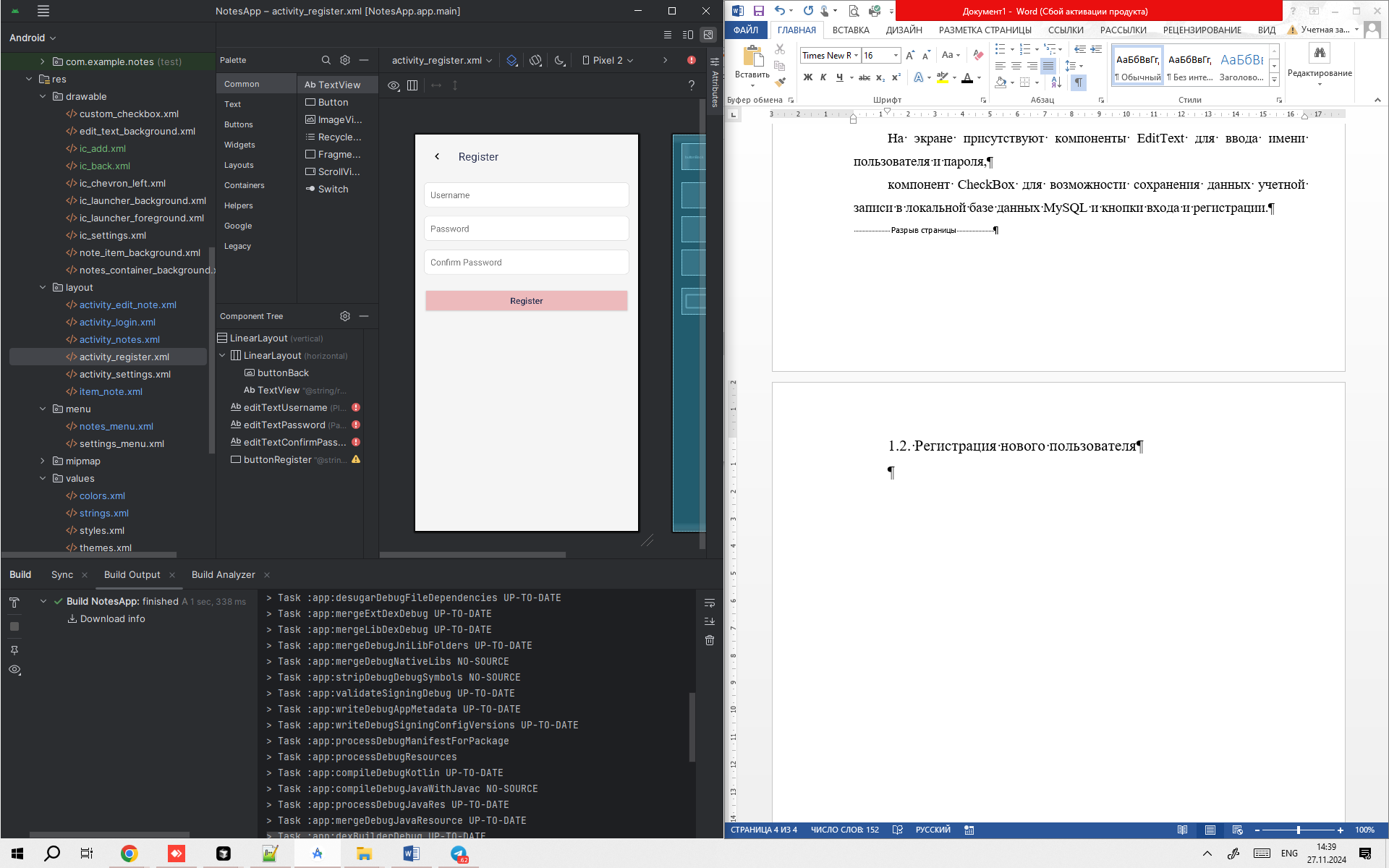


Рисунок 3 -- Activity для регистрации пользователя

На экране присутствуют компоненты EditText для ввода имени пользователя, пароля, а также подтверждение пароля,

Кнопки регистрации и возврата на окно входа.

1.3 Меню заметок

На рисунке 4 показан внешний вид формы меню заметок.

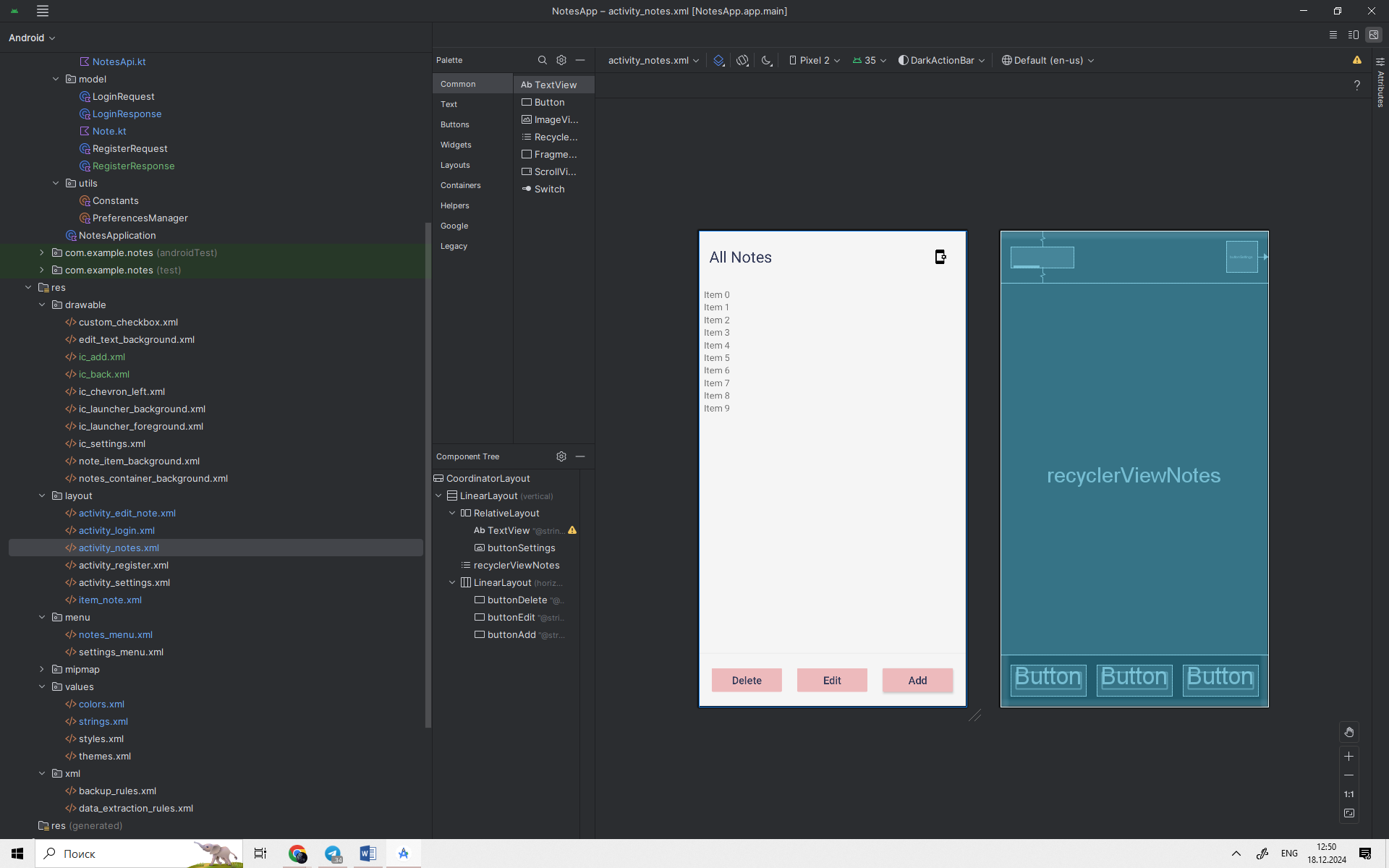


Рисунок 4 -- Activity меню заметок

На экране присутствуют компонент recyclerView для вывода заголовков заметок, кнопки для управления заметками (удалить, изменить, добавить).

А также кнопка для управления аккаунтом.

1.4 Заметка

На рисунке 5 показан внешний вид формы редактирования заметки.

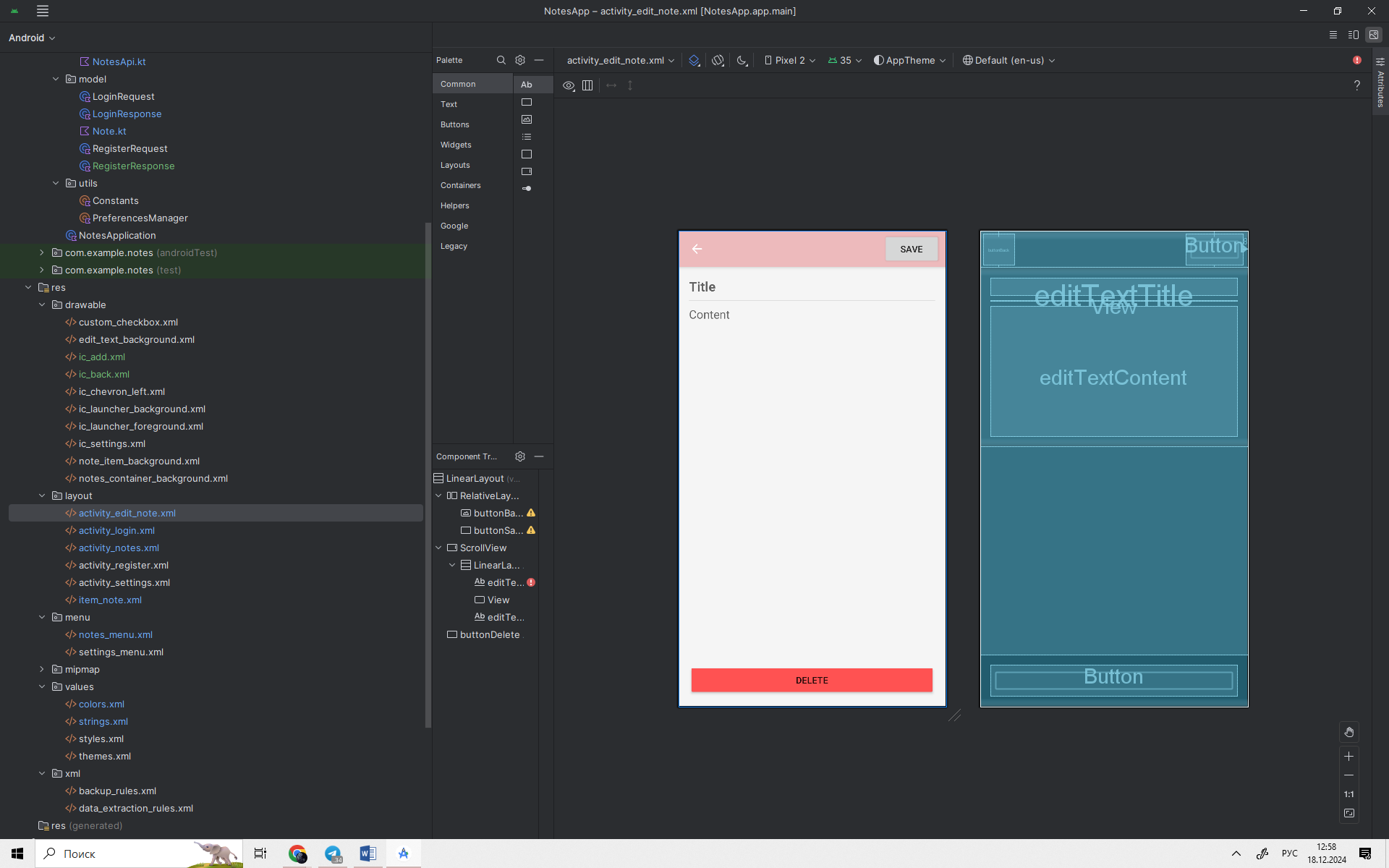


Рисунок 5 -- Activity редактирования заметки

На экране присутствуют компоненты EditText для ввода заголовка и текста заметки, кнопки для сохранения и удаления заметки.

2. Структура базы данных

База данных состоит из 2 таблиц:

1. account – список учетных записей.
2. notes – список заметок.

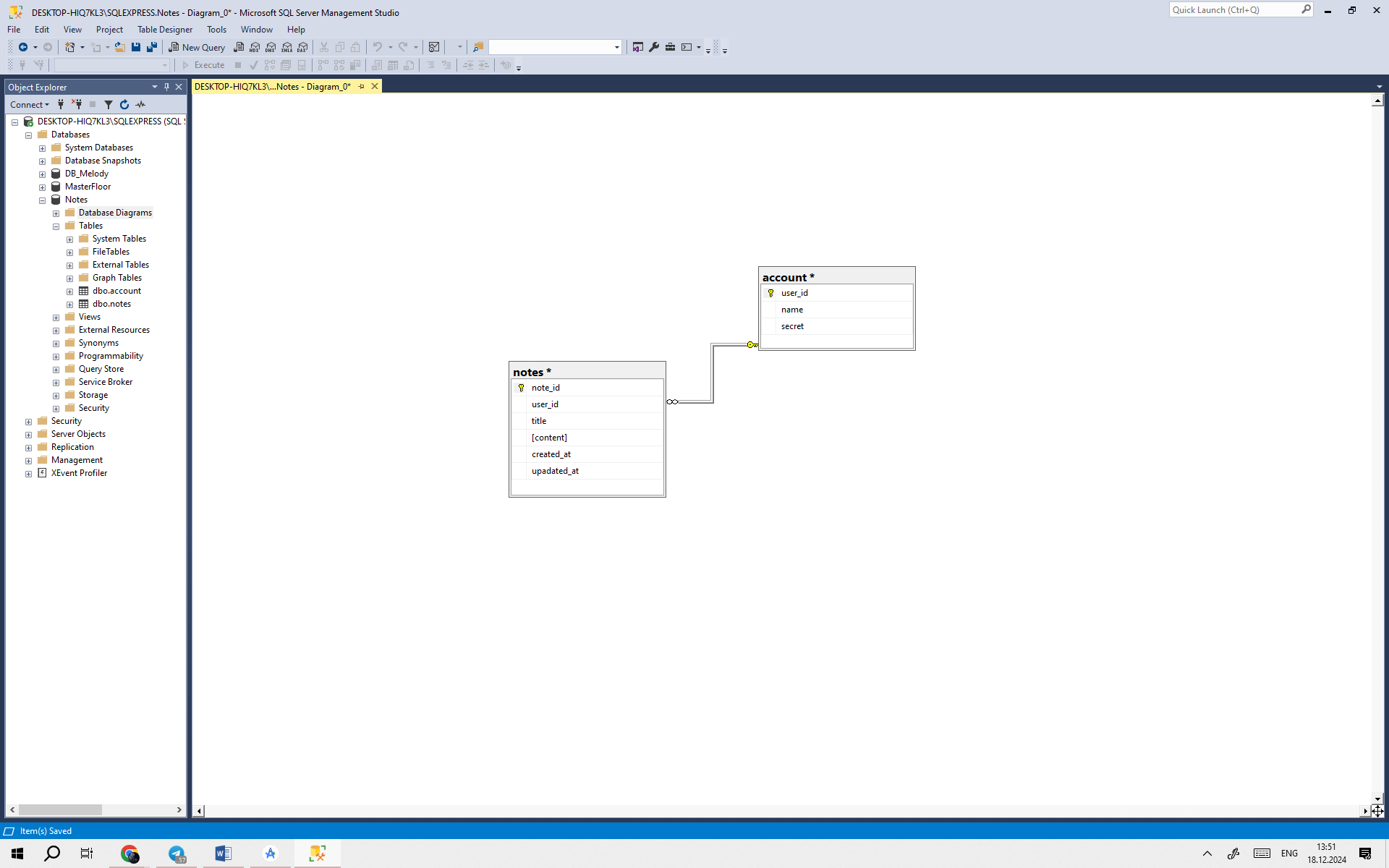


Рисунок 6 — ER-диаграмма базы данных

2.1 Таблица account

Содержит сведения об учетных записях пользователей. Таблица состоит из 3 столбцов:

1. user\_id – уникальный идентификатор пользователя.
2. name – логин пользователя.
3. secret – пароль учетной записи.

Подробной описание столбцов в таблице 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя | Тип | Размер | Уникальное | Пустое | Ключ |
| user\_id | Integer | - | да | нет | pk |
| name | Varchar | 50 | нет | - |
| secret | Varchar | 30 | - |

Таблица 1 – Описание типа данных столбцов

На рисунке 7 приведен пример данных, хранящихся в таблице.

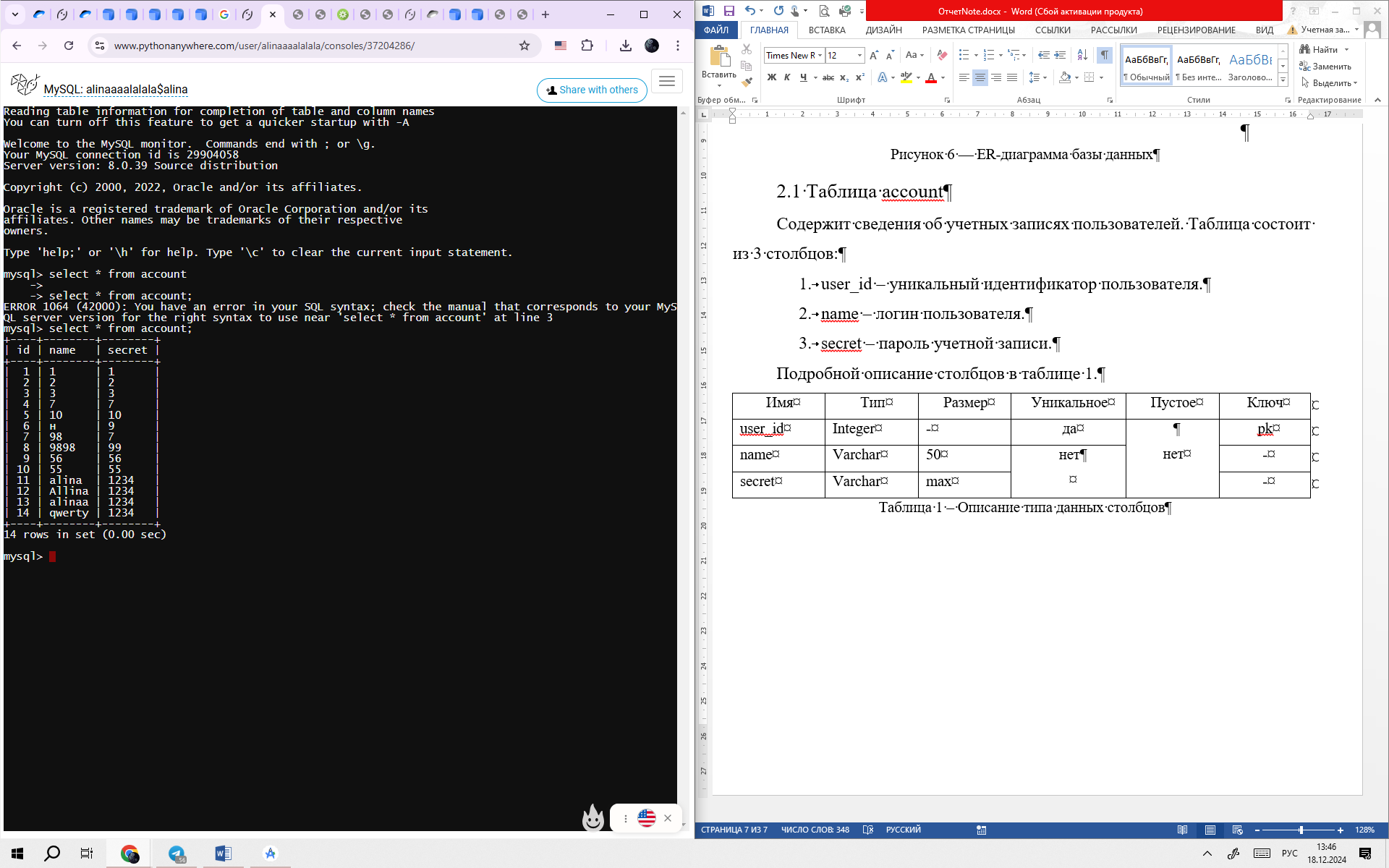


Рисунок 7 — Выборка данных из таблицы account

2.2 Таблица Notes

Содержит сведения о заметках. Таблица состоит из 6 столбцов:

1. id – уникальный идентификатор пользователя.
2. user\_id – логин пользователя.
3. title – заголовок заметки.
4. content – текст заметки.
5. created\_at – дата и время создания заметки.
6. updated\_at – дата и время последнего изменения заметки

Подробной описание столбцов в таблице 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя | Тип | Размер | Уникальное | Пустое | Ключ |
| id | Integer | - | да | нет | pk |
| user\_id | Integer | - | fk |
| title | Varchar | 50 | нет | - |
| content | Varchar | Max | - |
| created\_at | datatime | - | - |
| updated\_at | datatime | - | - |

Таблица 2 – Описание типа данных столбцов

На рисунке 8 приведен пример данных, хранящихся в таблице.

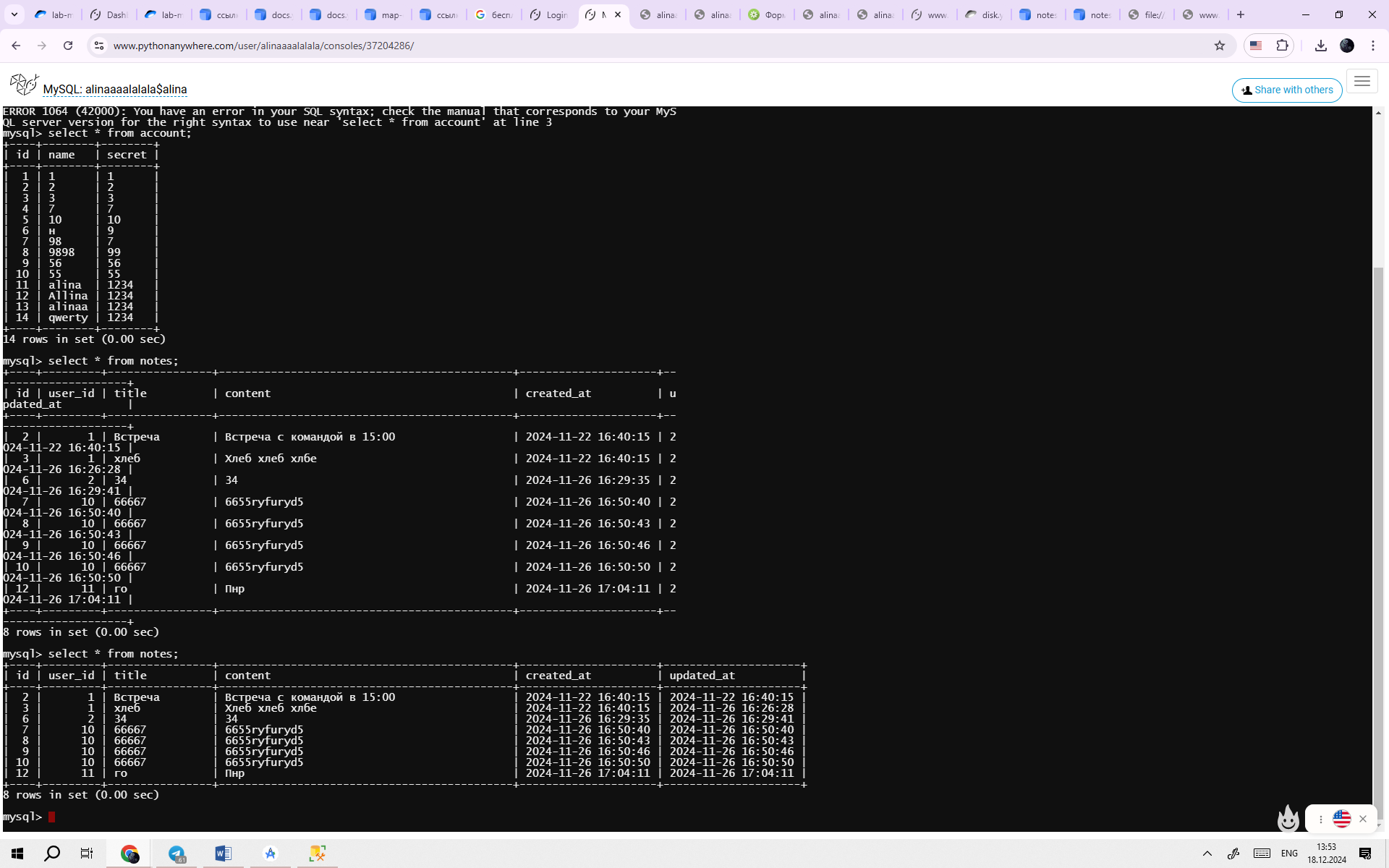


Рисунок 8 — Выборка данных из таблицы notes

3 Activity Login

class LoginActivity : AppCompatActivity() {  
 private lateinit var editTextUsername: EditText  
 private lateinit var editTextPassword: EditText  
 private lateinit var buttonLogin: Button  
 private lateinit var checkBoxRememberMe: CheckBox  
 private lateinit var textViewRegister: TextView  
 private val apiService by *lazy* { ApiClient.getApiService(this) }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_login*)  
  
 initViews()  
 setupClickListeners()  
 checkRememberedUser()  
 }  
  
 private fun initViews() {  
 editTextUsername = findViewById(R.id.*editTextUsername*)  
 editTextPassword = findViewById(R.id.*editTextPassword*)  
 buttonLogin = findViewById(R.id.*buttonLogin*)  
 checkBoxRememberMe = findViewById(R.id.*checkBoxRememberMe*)  
 textViewRegister = findViewById(R.id.*textViewRegister*)  
 }  
  
 private fun setupClickListeners() {  
 buttonLogin.setOnClickListener {  
 val username = editTextUsername.*text*.toString().*trim*()  
 val password = editTextPassword.*text*.toString().*trim*()  
  
 if (username.*isEmpty*() || password.*isEmpty*()) {  
 Toast.makeText(this, getString(R.string.*error\_empty\_fields*), Toast.*LENGTH\_SHORT*).show()  
 return@setOnClickListener  
 }  
  
 performLogin(username, password)  
 **}** textViewRegister.setOnClickListener **{** startActivity(Intent(this, RegisterActivity::class.*java*))  
 **}** }  
  
 private fun performLogin(username: String, password: String) {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
 val response = apiService.login(LoginRequest(username, password))  
 withContext(Dispatchers.Main) **{** if (response.*isSuccessful* && response.body() != null) {  
 val loginResponse = response.body()!!  
  
 if (loginResponse.status == "success") {  
 if (checkBoxRememberMe.*isChecked*) {  
 saveUserCredentials(username, password, loginResponse.userId ?: -1)  
 }  
  
 val intent = Intent(this@LoginActivity, NotesActivity::class.*java*)  
 intent.putExtra("userId", loginResponse.userId)  
 startActivity(intent)  
 finish()  
 } else {  
 Toast.makeText(  
 this@LoginActivity,  
 getString(R.string.*error\_login\_failed*),  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 } else {  
 Toast.makeText(  
 this@LoginActivity,  
 getString(R.string.*error\_login\_failed*),  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** } catch (e: Exception) {  
 withContext(Dispatchers.Main) **{** Toast.makeText(  
 this@LoginActivity,  
 getString(R.string.*error\_network*, e.message),  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** }  
 **}** }  
  
 private fun saveUserCredentials(username: String, password: String, userId: Long) {  
 getSharedPreferences("auth", *MODE\_PRIVATE*).edit().*apply* **{** putString("username", username)  
 putString("password", password)  
 putLong("userId", userId)  
 apply()  
 **}** }  
  
 private fun checkRememberedUser() {  
 val prefs = getSharedPreferences("auth", *MODE\_PRIVATE*)  
 val savedUsername = prefs.getString("username", null)  
 val savedPassword = prefs.getString("password", null)  
 val savedUserId = prefs.getLong("userId", -1)  
  
 if (savedUsername != null && savedPassword != null && savedUserId != -1L) {  
 performLogin(savedUsername, savedPassword)  
 }  
 }  
}

* 1. Activity Register

1. class RegisterActivity : AppCompatActivity() {  
    private lateinit var editTextUsername: EditText  
    private lateinit var editTextPassword: EditText  
    private lateinit var editTextConfirmPassword: EditText  
    private lateinit var buttonRegister: Button  
    private lateinit var buttonBack: ImageButton  
    private val apiService by *lazy* **{** ApiClient.getApiService(this) **}** override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.*activity\_register*)  
     
    initViews()  
    setupClickListeners()  
    }  
     
    private fun initViews() {  
    editTextUsername = findViewById(R.id.*editTextUsername*)  
    editTextPassword = findViewById(R.id.*editTextPassword*)  
    editTextConfirmPassword = findViewById(R.id.*editTextConfirmPassword*)  
    buttonRegister = findViewById(R.id.*buttonRegister*)  
    buttonBack = findViewById(R.id.*buttonBack*)  
    }  
     
    private fun setupClickListeners() {  
    buttonBack.setOnClickListener **{** finish()  
    **}** buttonRegister.setOnClickListener **{** val username = editTextUsername.*text*.toString().*trim*()  
    val password = editTextPassword.*text*.toString().*trim*()  
    val confirmPassword = editTextConfirmPassword.*text*.toString().*trim*()  
     
    when {  
    username.*isEmpty*() || password.*isEmpty*() || confirmPassword.*isEmpty*() -> {  
    Toast.makeText(this, getString(R.string.*error\_empty\_fields*), Toast.*LENGTH\_SHORT*)  
    .show()  
    }  
     
    password != confirmPassword -> {  
    Toast.makeText(  
    this,  
    getString(R.string.*error\_passwords\_dont\_match*),  
    Toast.*LENGTH\_SHORT* ).show()  
    }  
     
    else -> {  
    performRegistration(username, password)  
    }  
    }  
    **}** }  
     
    private fun performRegistration(username: String, password: String) {  
    *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
    val response = apiService.register(RegisterRequest(username, password))  
    withContext(Dispatchers.Main) **{** if (response.*isSuccessful* && response.body() != null) {  
    val registerResponse = response.body()!!  
    Log.d("RegisterActivity", "Response: $registerResponse")  
     
    if (registerResponse.status == "success") {  
    // Проверка наличия userId  
    val userId = registerResponse.userId  
    if (userId != null) {  
    Log.d("RegisterActivity", "User ID: $userId")  
     
    getSharedPreferences("auth", *MODE\_PRIVATE*).edit().*apply* {  
    putString("username", username)  
    putString("password", password)  
    putLong("userId", userId)  
    apply()  
    }  
     
    val intent =  
    Intent(this@RegisterActivity, NotesActivity::class.*java*)  
    intent.putExtra("userId", userId)  
    startActivity(intent)  
    finishAffinity()  
    } else {  
    Log.e("RegisterActivity", "User ID is null")  
    Toast.makeText(  
    this@RegisterActivity,  
    "Done",  
    Toast.*LENGTH\_SHORT* ).show()  
    }  
    } else {  
    Toast.makeText(  
    this@RegisterActivity,  
    registerResponse.message,  
    Toast.*LENGTH\_SHORT* ).show()  
    }  
    } else {  
    Toast.makeText(  
    this@RegisterActivity,  
    getString(R.string.*error\_register\_failed*),  
    Toast.*LENGTH\_SHORT* ).show()  
    }  
    **}** } catch (e: Exception) {  
    withContext(Dispatchers.Main) **{** Log.e("RegisterActivity", "Error: ${e.message}", e)  
    Toast.makeText(  
    this@RegisterActivity,  
    getString(R.string.*error\_network*, e.message),  
    Toast.*LENGTH\_SHORT* ).show()  
    **}** }  
    **}** }}

3.2 Activity Notes

package com.example.notes.activity  
  
import android.content.Intent  
import android.os.Bundle  
import android.util.Log  
import android.view.View  
import android.widget.Button  
import android.widget.ImageButton  
import android.widget.PopupMenu  
import android.widget.Toast  
import androidx.appcompat.app.AlertDialog  
import androidx.appcompat.app.AppCompatActivity  
import androidx.lifecycle.*lifecycleScope*import androidx.recyclerview.widget.LinearLayoutManager  
import androidx.recyclerview.widget.RecyclerView  
import com.google.android.material.floatingactionbutton.FloatingActionButton  
import com.example.notes.R  
import com.example.notes.adapter.NotesAdapter  
import com.example.notes.api.ApiClient  
import com.example.notes.model.Note  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class NotesActivity : AppCompatActivity() {  
 private val TAG = "NotesActivity"  
 private lateinit var recyclerView: RecyclerView  
 private lateinit var buttonAdd: Button  
 private lateinit var buttonDelete: Button  
 private lateinit var buttonEdit: Button  
 private lateinit var buttonSettings: ImageButton  
// private lateinit var fabAddNote: FloatingActionButton  
 private lateinit var notesAdapter: NotesAdapter  
 private var selectedNote: Note? = null  
 private var userId: String = ""  
 private val apiService by *lazy* { ApiClient.getApiService(this) }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_notes*)  
  
 userId = *intent*.getLongExtra("userId", -1).toString()  
 if (userId == "-1") {  
 Toast.makeText(this, getString(R.string.*error\_invalid\_user\_id*), Toast.*LENGTH\_SHORT*).show()  
 finish()  
 return  
 }  
  
 initViews()  
 setupRecyclerView()  
 setupClickListeners()  
 loadNotes()  
 }  
  
 private fun initViews() {  
 try {  
 recyclerView = findViewById(R.id.*recyclerViewNotes*)  
 buttonAdd = findViewById(R.id.*buttonAdd*)  
 buttonDelete = findViewById(R.id.*buttonDelete*)  
 buttonEdit = findViewById(R.id.*buttonEdit*)  
 buttonSettings = findViewById(R.id.*buttonSettings*)  
 // fabAddNote = findViewById(R.id.fabAddNote)  
  
 buttonDelete.*isEnabled* = false  
 buttonEdit.*isEnabled* = false  
 } catch (e: Exception) {  
 Log.e(TAG, "Error initializing views", e)  
 showToast("Error initializing app")  
 finish()  
 }  
 }  
  
 private fun setupRecyclerView() {  
 notesAdapter = NotesAdapter(*mutableListOf*()) **{** note **->** selectedNote = note  
 buttonDelete.*isEnabled* = true  
 buttonEdit.*isEnabled* = true  
 notesAdapter.setSelectedNote(note)  
 **}** recyclerView.*apply* {  
 *layoutManager* = LinearLayoutManager(this@NotesActivity)  
 *adapter* = notesAdapter  
 }  
 }  
  
 private fun setupClickListeners() {  
 buttonAdd.setOnClickListener {  
 Log.d(TAG, "Add button clicked")  
 openEditNote(null)  
 }  
  
 // fabAddNote.setOnClickListener {  
 // Log.d(TAG, "FAB clicked")  
 // openEditNote(null)  
 // }  
  
 buttonEdit.setOnClickListener {  
 Log.d(TAG, "Edit button clicked")  
 selectedNote?.*let* { note ->  
 Log.d(TAG, "Opening edit for note: ${note.id}")  
 openEditNote(note)  
 } ?: *run* {  
 Log.d(TAG, "No note selected for edit")  
 showToast(getString(R.string.*select\_note*))  
 }  
 }  
  
 buttonDelete.setOnClickListener {  
 selectedNote?.*let* { note **->** showDeleteConfirmationDialog(note)  
 } ?: showToast(getString(R.string.*select\_note*))  
 }  
  
 buttonSettings.setOnClickListener { view **->** showSettingsMenu(view)  
 }  
 }  
  
 private fun showDeleteConfirmationDialog(note: Note) {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*delete\_note\_title*))  
 .setMessage(getString(R.string.*delete\_note\_confirmation*))  
 .setPositiveButton(getString(R.string.*delete*)) { \_, \_ **->** deleteNote(note) }  
 .setNegativeButton(getString(R.string.*cancel*), null)  
 .show()  
 }  
  
 private fun showSettingsMenu(view: View) {  
 PopupMenu(this, view).*apply* {  
 *menuInflater*.inflate(R.menu.*notes\_menu*, *menu*)  
 setOnMenuItemClickListener { item **->** when (item.*itemId*) {  
 R.id.*menu\_server* -> {  
 startActivity(Intent(this@NotesActivity, SettingsActivity::class.*java*))  
 true  
 }  
 R.id.*menu\_logout* -> {  
 showLogoutConfirmationDialog()  
 true  
 }  
 else -> false  
 }  
 **}** show()  
 **}** }  
  
 private fun showLogoutConfirmationDialog() {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*logout*))  
 .setMessage(getString(R.string.*confirm\_logout*))  
 .setPositiveButton(getString(R.string.*yes*)) **{** \_, \_ **->** logout() **}** .setNegativeButton(getString(R.string.*no*), null)  
 .show()  
 }  
  
 private fun logout() {  
 getSharedPreferences("auth", *MODE\_PRIVATE*)  
 .edit()  
 .clear()  
 .apply()  
  
 val intent = Intent(this, LoginActivity::class.*java*)  
 intent.*flags* = Intent.*FLAG\_ACTIVITY\_NEW\_TASK* or Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK* startActivity(intent)  
 finish()  
 }  
  
 private fun openEditNote(note: Note?) {  
 try {  
 Log.d(TAG, "Opening EditActivity for note: ${note?.id}")  
 val intent = Intent(this, EditActivity::class.*java*).*apply* **{** putExtra("userId", userId)  
 if (note != null) {  
 putExtra(EditActivity.EXTRA\_NOTE\_ID, note.id.toString())  
 putExtra(EditActivity.EXTRA\_NOTE\_TITLE, note.title)  
 putExtra(EditActivity.EXTRA\_NOTE\_CONTENT, note.content)  
 }  
 **}** startActivityForResult(intent, if (note == null) REQUEST\_CREATE\_NOTE else REQUEST\_EDIT\_NOTE)  
 } catch (e: Exception) {  
 Log.e(TAG, "Error opening EditActivity", e)  
 showToast(getString(R.string.*error\_opening\_editor*))  
 }  
 }  
  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 Log.d(TAG, "onActivityResult: requestCode=$requestCode, resultCode=$resultCode")  
 if (resultCode == *RESULT\_OK*) {  
 when (requestCode) {  
 REQUEST\_CREATE\_NOTE, REQUEST\_EDIT\_NOTE -> {  
 loadNotes()  
 selectedNote = null  
 buttonDelete.*isEnabled* = false  
 buttonEdit.*isEnabled* = false  
 }  
 }  
 }  
 }  
  
 private fun deleteNote(note: Note) {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
 Log.d(TAG, "Deleting note: ${note.id}")  
 val response = apiService.deleteNote(note.id.toString())  
 withContext(Dispatchers.Main) **{** if (response.*isSuccessful*) {  
 notesAdapter.removeNote(note)  
 selectedNote = null  
 buttonDelete.*isEnabled* = false  
 buttonEdit.*isEnabled* = false  
 showToast(getString(R.string.*note\_deleted*))  
 } else {  
 showToast(getString(R.string.*error\_deleting\_note*))  
 }  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error deleting note", e)  
 withContext(Dispatchers.Main) **{** showToast(getString(R.string.*error\_network*, e.message))  
 **}** }  
 **}** }  
  
 private fun loadNotes() {  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
 Log.d(TAG, "Loading notes for user: $userId")  
 val response = apiService.getNotes(userId)  
 withContext(Dispatchers.Main) **{** if (response.*isSuccessful* && response.body() != null) {  
 val notesResponse = response.body()!!  
 Log.d(TAG, "Response received: $notesResponse")  
 if (notesResponse.status == "success") {  
 notesAdapter.updateNotes(notesResponse.notes)  
 Log.d(TAG, "Notes updated: ${notesResponse.notes}")  
 } else {  
 Log.e(TAG, "Error status: ${notesResponse.status}")  
 showToast(getString(R.string.*error\_loading\_notes*))  
 }  
 } else {  
 Log.e(TAG, "Response unsuccessful: ${response.code()}")  
 showToast(getString(R.string.*error\_loading\_notes*))  
 }  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error loading notes", e)  
 withContext(Dispatchers.Main) **{** showToast(getString(R.string.*error\_network*, e.message))  
 **}** }  
 **}** }  
  
 private fun showToast(message: String) {  
 Toast.makeText(this, message, Toast.*LENGTH\_SHORT*).show()  
 }  
  
 override fun onResume() {  
 super.onResume()  
 loadNotes()  
 }  
  
 companion object {  
 private const val REQUEST\_CREATE\_NOTE = 1  
 private const val REQUEST\_EDIT\_NOTE = 2  
 }  
}

3.3 Activity Edit

package com.example.notes.activity  
  
import android.os.Bundle  
import android.util.Log  
import android.widget.EditText  
import android.widget.ImageButton  
import android.widget.Button  
import android.widget.Toast  
import androidx.appcompat.app.AlertDialog  
import androidx.appcompat.app.AppCompatActivity  
import androidx.lifecycle.*lifecycleScope*import com.example.notes.R  
import com.example.notes.api.ApiClient  
import com.example.notes.model.CreateNoteRequest  
import com.example.notes.model.UpdateNoteRequest  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
class EditActivity : AppCompatActivity() {  
 private val TAG = "EditActivity"  
  
 private lateinit var editTextTitle: EditText  
 private lateinit var editTextContent: EditText  
 private lateinit var buttonBack: ImageButton  
 private lateinit var buttonSave: Button  
 private lateinit var buttonDelete: Button  
  
 private var noteId: String? = null  
 private var userId: String = ""  
 private var originalTitle: String = ""  
 private var originalContent: String = ""  
 private val apiService by *lazy* **{** ApiClient.getApiService(this) **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 try {  
 setContentView(R.layout.*activity\_edit\_note*)  
 Log.d(TAG, "EditActivity created")  
  
 // Получаем userId из Intent  
 userId = *intent*.getStringExtra("userId") ?: *run* **{** Log.e(TAG, "Invalid userId")  
 Toast.makeText(this, getString(R.string.*error\_invalid\_user\_id*), Toast.*LENGTH\_SHORT*).show()  
 finish()  
 return  
 **}** initViews()  
 loadNoteData()  
 setupClickListeners()  
 } catch (e: Exception) {  
 Log.e(TAG, "Error in onCreate", e)  
 Toast.makeText(this, "Error initializing editor", Toast.*LENGTH\_SHORT*).show()  
 finish()  
 }  
 }  
  
 private fun initViews() {  
 try {  
 editTextTitle = findViewById(R.id.*editTextTitle*)  
 editTextContent = findViewById(R.id.*editTextContent*)  
 buttonBack = findViewById(R.id.*buttonBack*)  
 buttonSave = findViewById(R.id.*buttonSave*)  
 buttonDelete = findViewById(R.id.*buttonDelete*)  
 } catch (e: Exception) {  
 Log.e(TAG, "Error initializing views", e)  
 throw e  
 }  
 }  
  
 private fun loadNoteData() {  
 try {  
 noteId = *intent*.getStringExtra(EXTRA\_NOTE\_ID)  
 Log.d(TAG, "Loading note data. NoteId: $noteId")  
  
 if (noteId != null) {  
 // Редактирование существующей заметки  
 originalTitle = *intent*.getStringExtra(EXTRA\_NOTE\_TITLE) ?: ""  
 originalContent = *intent*.getStringExtra(EXTRA\_NOTE\_CONTENT) ?: ""  
  
 editTextTitle.setText(originalTitle)  
 editTextContent.setText(originalContent)  
 Log.d(TAG, "Loaded existing note: title=$originalTitle")  
 } else {  
 // Новая заметка - скрываем кнопку удаления  
 buttonDelete.*visibility* = android.view.View.*GONE* Log.d(TAG, "Creating new note")  
 }  
 } catch (e: Exception) {  
 Log.e(TAG, "Error loading note data", e)  
 Toast.makeText(this, "Error loading note", Toast.*LENGTH\_SHORT*).show()  
 finish()  
 }  
 }  
  
 private fun setupClickListeners() {  
 buttonBack.setOnClickListener **{** checkUnsavedChanges()  
 **}** buttonSave.setOnClickListener **{** saveNote()  
 **}** buttonDelete.setOnClickListener **{** showDeleteConfirmationDialog()  
 **}** }  
  
 private fun checkUnsavedChanges() {  
 val currentTitle = editTextTitle.*text*.toString().*trim*()  
 val currentContent = editTextContent.*text*.toString().*trim*()  
  
 if (currentTitle != originalTitle || currentContent != originalContent) {  
 showUnsavedChangesDialog()  
 } else {  
 finish()  
 }  
 }  
  
 private fun showUnsavedChangesDialog() {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*discard\_changes\_title*))  
 .setMessage(getString(R.string.*discard\_changes\_message*))  
 .setPositiveButton(getString(R.string.*yes*)) **{** \_, \_ **->** finish() **}** .setNegativeButton(getString(R.string.*no*), null)  
 .show()  
 }  
  
 private fun showDeleteConfirmationDialog() {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*delete\_note\_title*))  
 .setMessage(getString(R.string.*delete\_note\_confirmation*))  
 .setPositiveButton(getString(R.string.*delete*)) **{** \_, \_ **->** deleteNote() **}** .setNegativeButton(getString(R.string.*cancel*), null)  
 .show()  
 }  
  
 private fun saveNote() {  
 val title = editTextTitle.*text*.toString().*trim*()  
 val content = editTextContent.*text*.toString().*trim*()  
  
 if (title.*isEmpty*() || content.*isEmpty*()) {  
 Toast.makeText(this, getString(R.string.*error\_empty\_fields*), Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
 Log.d(TAG, "Saving note: title=$title")  
 val response = if (noteId == null) {  
 // Создание новой заметки  
 apiService.createNote(CreateNoteRequest(userId, title, content))  
 } else {  
 // Обновление существующей заметки  
 apiService.updateNote(noteId!!, UpdateNoteRequest(title, content))  
 }  
  
 withContext(Dispatchers.Main) **{** if (response.*isSuccessful*) {  
 setResult(*RESULT\_OK*)  
 Toast.makeText(  
 this@EditActivity,  
 getString(if (noteId == null) R.string.*note\_created* else R.string.*note\_updated*),  
 Toast.*LENGTH\_SHORT* ).show()  
 finish()  
 } else {  
 Toast.makeText(  
 this@EditActivity,  
 getString(R.string.*error\_saving\_note*),  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error saving note", e)  
 withContext(Dispatchers.Main) **{** Toast.makeText(  
 this@EditActivity,  
 getString(R.string.*error\_network*, e.message),  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** }  
 **}** }  
  
 private fun deleteNote() {  
 if (noteId == null) return  
  
 *lifecycleScope*.*launch*(Dispatchers.IO) **{** try {  
 Log.d(TAG, "Deleting note: $noteId")  
 val response = apiService.deleteNote(noteId!!)  
 withContext(Dispatchers.Main) **{** if (response.*isSuccessful*) {  
 setResult(*RESULT\_OK*)  
 Toast.makeText(  
 this@EditActivity,  
 getString(R.string.*note\_deleted*),  
 Toast.*LENGTH\_SHORT* ).show()  
 finish()  
 } else {  
 Toast.makeText(  
 this@EditActivity,  
 getString(R.string.*error\_deleting\_note*),  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error deleting note", e)  
 withContext(Dispatchers.Main) **{** Toast.makeText(  
 this@EditActivity,  
 getString(R.string.*error\_network*, e.message),  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** }  
 **}** }  
  
 override fun onBackPressed() {  
 checkUnsavedChanges()  
 }  
  
 companion object {  
 const val EXTRA\_NOTE\_ID = "extra\_note\_id"  
 const val EXTRA\_NOTE\_TITLE = "extra\_note\_title"  
 const val EXTRA\_NOTE\_CONTENT = "extra\_note\_content"  
 }  
}