ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

ассистент		И.Д. Свеженин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6

СУБД и использование сетевых сервисов

по курсу: Кроссплатформенное программирование

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4329		Д.С. Шаповалова
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1. Цель работы:

Выполнить проектирование и разработку мобильного приложения под ОС Android на языке программирования высокого уровня Kotlin.

2. Задание:

1. Задание «БД»

1) Создать БД в соответствии с вариантом (предметной областью) – для определенного класса;
2) Реализовать добавление данных в БД с помощью отдельного Activity;
3) Вывести данные из БД в RecyclerView. Также использовать фрагменты.
4) Обработка долгого нажатия на элемент списка;
5) Создание диалогового окна для выбора «Просмотр», «Удаление», «Обновление»;
6) Обработка нажатия элементов диалогового окна, например, для подтверждения удаления или обновления;
7) Реализовать обновление данных в БД с помощью отдельного Activity;
8) Реализовать удаление данных из БД с помощью отдельного Activity;

При работе с БД использовать библиотеку Room.

2. Задание «JSON»

1) Скачать JSON из интернета (HttpURLConnection/Retrofit);
2) Распарсить JSON;
3) Использовать Thread для работы с JSON.

Вариант 24

Предметная область - Попугаи.

3. Краткое описание хода разработки, алгоритма работы программы и назначение используемых технологий

Ход разработки:

1. Разработать главную активность
2. Реализация базы данных: Подключение библиотеки **Room**, создание сущности Parrot, DAO-интерфейса и самой базы данных.
3. Реализация навигации: Настройка BottomNavigationView и создание фрагментов (HomeFragment, ParrotsListFragment, SettingsFragment) для навигации.
4. Создание CRUD-функционала: Разработка отдельных активити для добавления (AddParrotActivity), просмотра (ViewParrotActivity), обновления (UpdateParrotActivity) и удаления (DeleteParrotActivity) данных.
5. Интеграция списка: Подключение RecyclerView и создание адаптера (ParrotAdapter) для отображения списка попугаев из базы данных.
6. Обработка событий: Реализация диалогового окна при долгом нажатии на элемент списка с выбором действий.
7. Работа с JSON: Реализация загрузки JSON из интернета с использованием HttpURLConnection, парсинга с помощью Gson и выполнения операций в отдельном потоке.

Используемые технологии:

- Kotlin: язык программирования, для написания логики приложения.
- Room: библиотека для удобной работы с локальной SQLite-базой данных, обеспечивает типобезопасность и упрощает написание SQL-запросов.
- RecyclerView: компонент для эффективного отображения больших списков данных с возможностью повторного использования элементов интерфейса.
- BottomNavigationView: компонент для реализации навигации между фрагментами снизу экрана.
- Coroutines: позволяет выполнять асинхронные операции (например, работу с БД или сетью) без блокировки основного потока.
- HttpURLConnection / Gson: используются для загрузки JSON-данных из интернета и их преобразования в Kotlin-объекты.
- Material Design: набор гайдлайнов и компонентов для создания современного и интуитивно понятного интерфейса.

Описание структуры работы частей приложения:

1. Основная активность (MainActivity)

MainActivity является точкой входа приложения и отвечает за:

- Инициализацию пользовательского интерфейса: Загрузка основной разметки (activity_main.xml), содержащей FrameLayout (для размещения фрагментов) и BottomNavigationView (для навигации).
- Настройку навигации: Обработка нажатий на элементы BottomNavigationView и замена текущего фрагмента в FrameLayout на соответствующий.
- Запуск загрузки данных: Асинхронная инициализация загрузки JSON из интернета при запуске приложения.

2. Фрагменты (HomeFragment, ParrotsListFragment, SettingsFragment)

Фрагменты представляют собой отдельные экраны, которые заменяются в FrameLayout при навигации:

- HomeFragment: Отображает приветственное сообщение или стартовую информацию.
- ParrotsListFragment: Основной фрагмент, отвечающий за отображение списка попугаев. Содержит:
  - RecyclerView — для отображения списка.
  - ParrotAdapter — связывает данные из базы с элементами списка.
  - Кнопку «Добавить» — открывает AddParrotActivity.
  - Обработчик долгого нажатия — вызывает диалог с действиями.
- SettingsFragment: Отображает информацию о настройках или пустой экран.

3. База данных (Room)

- Parrot.kt (Entity): Определяет структуру таблицы parrots в базе данных (поля: id, name, species, age).
- ParrotDao.kt (Data Access Object): Содержит методы для выполнения SQL-запросов: получение, добавление, обновление, удаление записей.
- ParrotDatabase.kt: Основной класс базы данных, реализует паттерн Singleton, обеспечивает доступ к DAO.

4. Активити для CRUD-операций

Каждая активити отвечает за один тип операции с базой данных:

- AddParrotActivity: Позволяет пользователю ввести данные нового попугая и сохранить его в базу данных.

- ViewParrotActivity: Отображает подробную информацию о выбранном попугае.
- UpdateParrotActivity: Позволяет редактировать данные существующего попугая.
- DeleteParrotActivity: Открывает диалог подтверждения и удаляет попугая из базы данных.

5. Отображение списка (RecyclerView и ParrotAdapter)

- RecyclerView: Компонент, отображающий список элементов в ParrotsListFragment.
- ParrotAdapter: Адаптер, который:
  - Создаёт и связывает элементы интерфейса (ViewHolder) с данными из списка попугаев.
  - Отвечает за отображение имени, вида и возраста попугая в item_parrot.xml.
  - Обрабатывает клики и долгие нажатия, передавая информацию в ParrotsListFragment.

6. Обработка событий и диалоги

- Диалог при долгом нажатии: При удержании элемента в RecyclerView открывается AlertDialog с тремя действиями: «Просмотр», «Удалить», «Обновить».
- Переход между активити: В зависимости от выбранного действия, запускается соответствующая активити (ViewParrotActivity, DeleteParrotActivity, UpdateParrotActivity).

7. Работа с JSON

- JsonHelper.kt: Отдельный объект, отвечающий за:
  - Загрузку JSON-данных с удалённого сервера с помощью HttpURLConnection.
  - Парсинг JSON в Kotlin-объекты с помощью библиотеки Gson.
  - Обработку ошибок (например, FileNotFoundException, JsonSyntaxException).
- Загрузка в отдельном потоке: Использование Thread или Coroutines для выполнения сетевых операций вне основного потока, чтобы не блокировать UI.
- Проверка на дубликаты: Перед добавлением данных в базу, проверяется, существует ли уже попугай с таким именем, чтобы избежать дублирования.

8. Взаимодействие компонентов

- MainActivity управляет навигацией между фрагментами.
- ParrotsListFragment подписывается на изменения в базе данных через Flow и обновляет список в RecyclerView.
- При добавлении, обновлении или удалении данных в одной из CRUD-активити, изменения автоматически отражаются в списке на экране ParrotsListFragment.
- Загрузка JSON из интернета происходит асинхронно и безопасно добавляется в базу данных, если данные отсутствуют.

Алгоритм работы программы:

1) При запуске приложения загружается MainActivity, инициализируется навигация и запускается асинхронная загрузка JSON.
2) Загруженные данные JSON проверяются на дубликаты и, при отсутствии, добавляются в базу данных.
3) Пользователь может переключаться между фрагментами с помощью BottomNavigationView.
4) В фрагменте ParrotsListFragment отображается список попугаев из базы данных. Список обновляется автоматически при изменении данных.
5) При нажатии на кнопку «Добавить» открывается AddParrotActivity, где пользователь может ввести данные и сохранить нового попугая.
6) При долгом нажатии на элемент списка открывается диалог с действиями: «Просмотр», «Удалить», «Обновить». Выбор действия приводит к открытию соответствующей активности.
7) Все операции с базой данных выполняются асинхронно, чтобы не блокировать пользовательский интерфейс.

4. Скриншоты, иллюстрирующие результаты работы программы:

В качестве демонстрации работы программы приведены несколько скриншотов каждого фрагмента.

Рисунок 2.1 – Стартовый экран

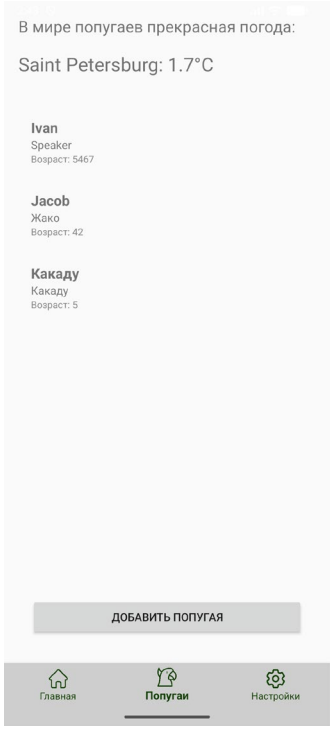

Рисунок 3.1 – Список попугаев

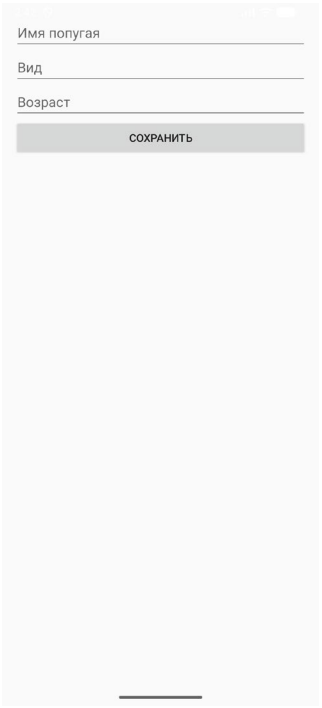

Рисунок 3.2 – Добавить попугая

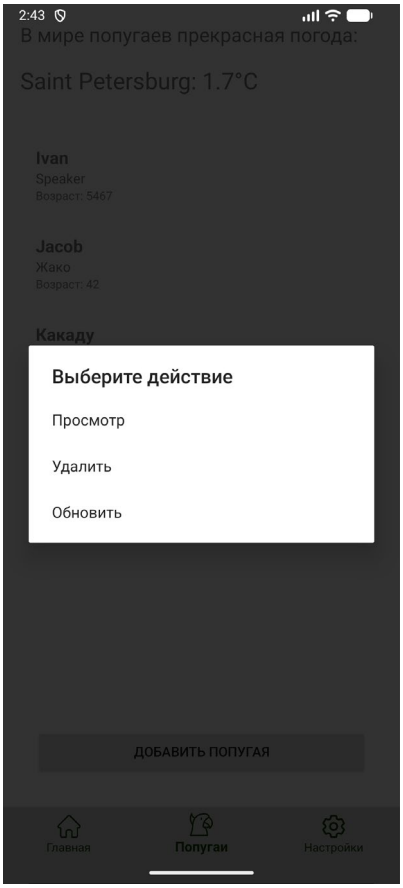

Рисунок 3.3 – При долгом нажатии на попугая

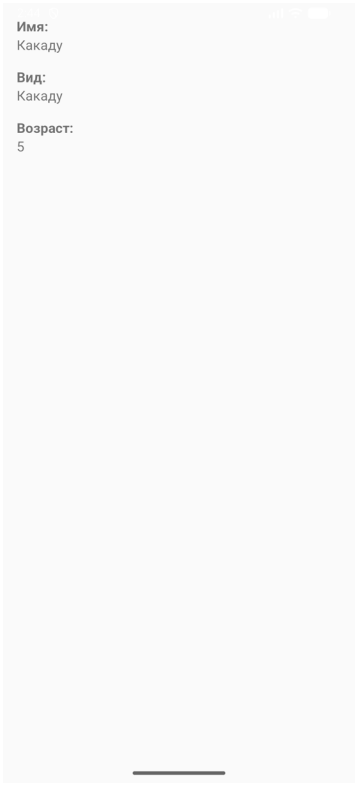

Рисунок 3.4 – Просмотр попугая

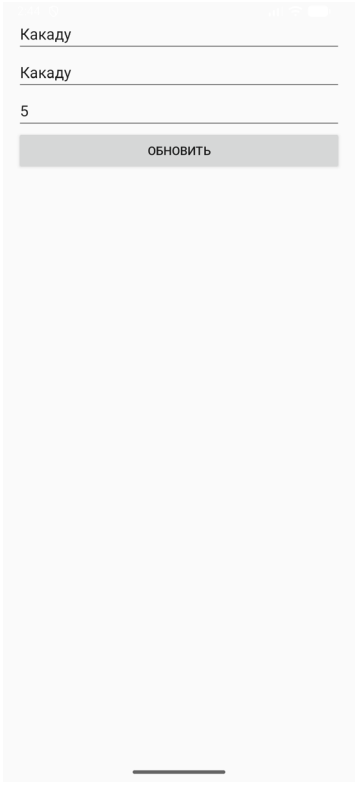

Рисунок 3.5 – Обновить попугая

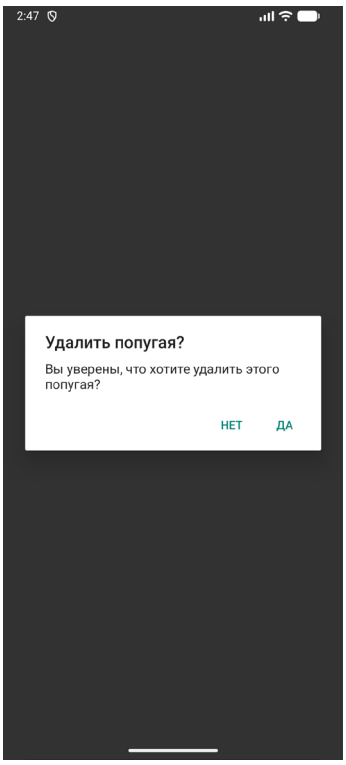

Рисунок 3.6 – Удалить попугая

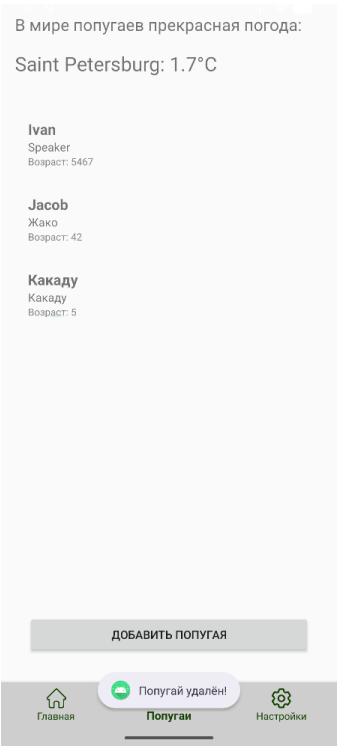

Рисунок 3.7 – Удалён попугай

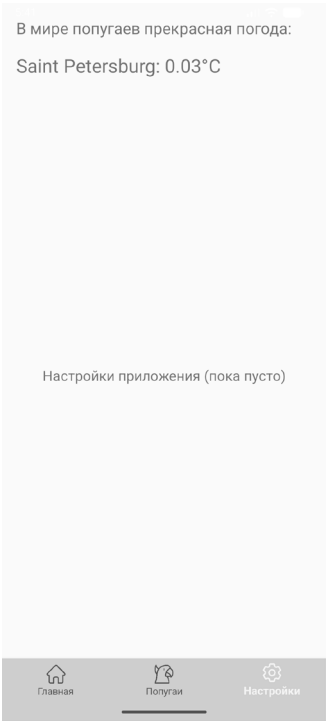

Рисунок 4 – Пустая вкладка настроек

Рисунок 5 – Результат работы API, при отсутствии интернета

5. Вывод:

В ходе выполнения работы было разработано мобильное Android-приложение на языке Kotlin, темой которого является попугаи. Приложение реализует функционал хранения, добавления, просмотра, обновления и удаления данных о попугаях с использованием локальной базы данных Room, а также получение и парсинг JSON из интернета в отдельном потоке. Реализована навигация по фрагментам с помощью BottomNavigationView, а также отображение данных в списке через RecyclerView с возможностью обработки долгого нажатия и вызова диалогового окна с действиями.

Проект успешно демонстрирует современный подход к разработке Android-приложений, включающий работу с асинхронными операциями, безопасное хранение данных, корректную обработку ошибок и удобный пользовательский интерфейс. Все поставленные задачи были выполнены: создана база данных, реализован CRUD-функционал, интегрирована работа с JSON, настроена навигация и отображение данных. Приложение стабильно работает, корректно обрабатывает дубликаты данных и обеспечивает удобное взаимодействие с пользователем.

Разработка позволила получить практические навыки работы с такими технологиями, как Room, RecyclerView, Coroutines, Gson, Navigation, и закрепить знания по архитектуре и жизненному циклу Android-приложений.

ПРИЛОЖЕНИЕ А

Листинг программы:

Полностью весь код представлен по ссылке: MyataEtoki/CP-6-lr: КП 6 лаба

MainActivity.kt:

```
package com.example.a6lr

import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.net.ConnectivityManager
import android.os.Build
import android.os.Bundle
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import com.google.android.material.bottomnavigation.BottomNavigationView
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class MainActivity : AppCompatActivity() {

    private val REQUEST_CODE_PERMISSIONS = 100

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

        // Проверяем разрешения
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.INTERNET) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.INTERNET), REQUEST_CODE_PERMISSIONS)
        }

        // Запускаем API-запрос
        fetchWeatherData()

        // Загружаем JSON в отдельном потоке и добавляем в БД
        loadJsonToDatabase()

        // Устанавливаем навигацию меню
        setupNavigation()
    }

    // Работа с API - вызов данных о температуре в городе N
    private fun fetchWeatherData() {
        val retrofit = Retrofit.Builder()
            .baseUrl("https://api.openweathermap.org/data/2.5/") // Убраны
лишние пробелы
            .addConverterFactory(GsonConverterFactory.create())
            .build()
```

```
        val service = retrofit.create(WeatherApiService::class.java)

        // запрос
        val call = service.getCurrentWeather("Saint Petersburg",
"0afc736cf6ba687d3e6d20ed62574b95", "metric")

        // отправляем запрос
        call.enqueue(object : Callback<WeatherResponse> {
            // сервер ответил, парсим ответ
            override fun onResponse(call: Call<WeatherResponse>, response:
Response<WeatherResponse>) {
                if (response.isSuccessful) {
                    val weather = response.body()
                    findViewById<TextView>(R.id.textView).text =
"${weather?.name}: ${weather?.main?.temp}°C"
                } else {
                    Toast.makeText(this@MainActivity, "Ошибка:
${response.code()}", Toast.LENGTH_SHORT).show()
                }
            }
            // сервер не ответил
            override fun onFailure(call: Call<WeatherResponse>, t: Throwable)
{
                if (!isNetworkAvailable()) {
                    Toast.makeText(this@MainActivity, "Нет подключения к
интернету", Toast.LENGTH_LONG).show()
                } else {
                    Toast.makeText(this@MainActivity, "Ошибка сети:
${t.message}", Toast.LENGTH_SHORT).show()
                }
            }
        })
    }

    // Загрузка JSON в отдельном потоке и добавление в БД
    private fun loadJsonToDatabase() {
        CoroutineScope(Dispatchers.IO).launch {
            val jsonList = JsonHelper.fetchAndParseJson()
            jsonList?.forEach { jsonParrot ->
                val database = ParrotDatabase.getDatabase(applicationContext)
                val existingParrot =
database.parrotDao().getParrotByName(jsonParrot.name)
                if (existingParrot == null) {
                    database.parrotDao().insertParrot(
                        Parrot(name = jsonParrot.name, species =
jsonParrot.species, age = jsonParrot.age)
                    )
                }
            }
        }
    }

    // Проверка на наличие интернета
    private fun isNetworkAvailable(): Boolean {
        val connectivityManager =
getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

        return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            val network = connectivityManager.activeNetwork
            val capabilities =
connectivityManager.getNetworkCapabilities(network)
```

```
capabilities?.hasCapability(android.net.NetworkCapabilities.NET_CAPABILITY_IN
TERNET) == true
        } else {
            @Suppress("DEPRECATION")
            val networkInfo = connectivityManager.activeNetworkInfo
            networkInfo?.isConnected == true
        }
    }

    // Навигация нижнего меню - замена фрагментов
    private fun setupNavigation() {
        val bottomNav =
findViewById<BottomNavigationView>(R.id.bottom_navigation)

        bottomNav.setOnItemSelectedListener { item ->
            when (item.itemId) {
                R.id.nav_home -> {
                    replaceFragment(HomeFragment())
                    true
                }
                R.id.nav_parrots -> {
                    replaceFragment(ParrotsListFragment())
                    true
                }
                R.id.nav_settings -> {
                    replaceFragment(SettingsFragment())
                    true
                }
                else -> false
            }
        }

        // Загрузка начального фрагмента
        replaceFragment(HomeFragment())
    }

    // Замена фрагмента через их менеджер
    private fun replaceFragment(fragment: Fragment) {
        supportFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, fragment)
            .commit()
    }
}
```