Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования

«Национальный исследовательский университет ИТМО»

Факультет прикладной информатики

Отчет о курсовой работе
по дисциплине “Мобильная разработка (Android и IOs)”
на тему “Разработка мобильного приложения Мессенджер”

Выполнила:

студентка группы K3442
Смирнова Глафира Денисовна

Проверил:

преподаватель ФПИН ИТМО
Шатинский Григорий Сергеевич

Санкт-Петербург, 2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время мобильные приложения являются неотъемлемой частью повседневной жизни и широко используются для общения, обмена информацией и организации взаимодействия между пользователями. Особое место среди них занимают мессенджеры, которые предъявляют высокие требования к удобству интерфейса, стабильности работы, архитектуре приложения и корректной обработке данных как в онлайн-, так и в офлайн-режиме.

Целью данной курсовой работы является поэтапная разработка Android-приложения «Мессенджер» с использованием современных средств и подходов мобильной разработки. В рамках работы последовательно реализовывались базовая навигация и структура приложения, архитектура MVVM, работа с сетевыми источниками данных и локальным хранилищем, а также элементы интерактивного пользовательского интерфейса и фоновой обработки данных.

В ходе выполнения работы особое внимание уделялось разделению ответственности между слоями приложения, использованию рекомендованных библиотек Android Jetpack, обработке жизненного цикла компонентов, а также повышению устойчивости приложения к изменению состояния системы, таким как отсутствие сетевого соединения или сворачивание приложения.

Ход работы

В рамках серии лабораторных работ была выполнена поэтапная разработка Android-приложения «Мессенджер». Разработка велась с постепенным усложнением архитектуры, пользовательского интерфейса и логики работы с данными.

Лабораторная работа 1

Тема: “Разработка базового Android-приложения с навигацией”.

Задачи:

1) создать Android-приложение с одной главной активностью;

2) реализовать навигацию между экранами с помощью Bottom Navigation;

3) добавить три экрана:

- экран новостной ленты (заглушка),
- экран профиля пользователя,
- экран настроек;

4) использовать Navigation Component (NavController и navigation graph);

5) реализовать логирование жизненного цикла Activity и Fragment;

6) настроить структуру проекта и .gitignore;

7) обеспечить корректный запуск приложения без ошибок.

Для реализации была создана главная активность MainActivity (см. Листинг 1), которая выступает точкой входа в приложение. Навигация между экранами (см. Листинг 2) организована с использованием NavHostFragment и BottomNavigationView (см. Рисунок 1).

Для каждого экрана был создан отдельный Fragment (см. Листинги 3-5):

- FeedFragment - экран ленты (на данном этапе заглушка),
- ProfileFragment - экран профиля пользователя,
- SettingsFragment - экран настроек.

Навигация между фрагментами описана в navigation graph. В каждом фрагменте и в активности были переопределены методы жизненного цикла (onCreate, onStart, onResume, onDestroy и др.) с выводом сообщений в Logcat (см. Рисунок 2).

Также был настроен файл .gitignore, исключающий системные и временные файлы Android Studio, и подготовлен APK-файл для проверки работы приложения.

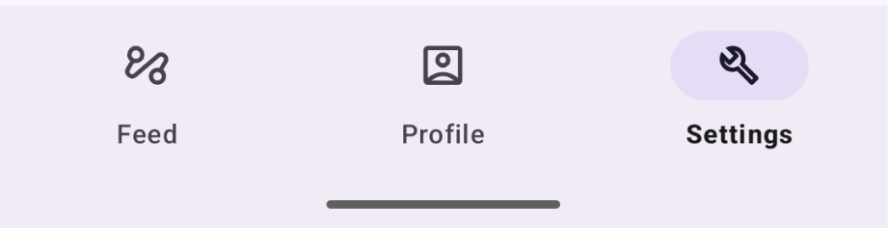

Рисунок 1 - Bottom Navigation

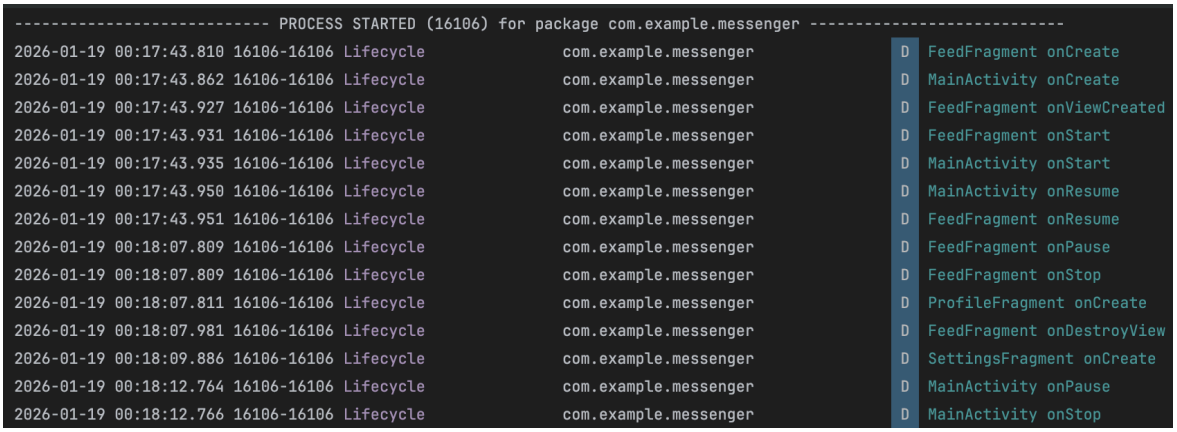

Рисунок 2 - Logcat с сообщениями жизненного цикла

Листинг 1 - MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        val isDark = getSharedPreferences("settings", MODE_PRIVATE)
            .getBoolean("dark_theme", false)
        AppCompatDelegate.setDefaultNightMode(
            if (isDark) AppCompatDelegate.MODE_NIGHT_YES
            else AppCompatDelegate.MODE_NIGHT_NO
        )

        val navHost = supportFragmentManager
```

```
            .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
        val navController = navHost.navController
        val bottom = findViewById<BottomNavigationView>(R.id.bottom_nav)
        bottom.setupWithNavController(navController)

        Log.d("Lifecycle", "MainActivity onCreate")
    }

    override fun onStart() { super.onStart(); Log.d("Lifecycle",
"MainActivity onStart") }
    override fun onResume() { super.onResume(); Log.d("Lifecycle",
"MainActivity onResume") }
    override fun onPause() { Log.d("Lifecycle", "MainActivity onPause");
super.onPause() }
    override fun onStop() { Log.d("Lifecycle", "MainActivity onStop");
super.onStop() }
    override fun onDestroy() { Log.d("Lifecycle", "MainActivity onDestroy");
super.onDestroy() }
}
```

Листинг 2 - Код navigation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    app:startDestination="@id/feedFragment">

    <fragment
        android:id="@+id/feedFragment"
        android:name="com.example.messenger.ui.feed.FeedFragment"
        android:label="Feed"
        tools:layout="@layout/fragment_feed" />

    <fragment
        android:id="@+id/profileFragment"
        android:name="com.example.messenger.ui.profile.ProfileFragment"
        android:label="Profile"
        tools:layout="@layout/fragment_profile" />

    <fragment
        android:id="@+id/settingsFragment"
        android:name="com.example.messenger.ui.settings.SettingsFragment"
        android:label="Settings"
        tools:layout="@layout/fragment_settings" />
</navigation>
```

Листинг 3 - FeedFragment

```
class FeedFragment : Fragment() {
```

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

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d("Lifecycle", "FeedFragment onCreate")
    }

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        Log.d("Lifecycle", "FeedFragment onViewCreated")
    }

    override fun onStart() { super.onStart(); Log.d("Lifecycle",
"FeedFragment onStart") }
    override fun onResume() { super.onResume(); Log.d("Lifecycle",
"FeedFragment onResume") }
    override fun onPause() { Log.d("Lifecycle", "FeedFragment onPause");
super.onPause() }
    override fun onStop() { Log.d("Lifecycle", "FeedFragment onStop");
super.onStop() }

    override fun onDestroyView() {
        Log.d("Lifecycle", "FeedFragment onDestroyView")
        _binding = null
        super.onDestroyView()
    }

    override fun onDestroy() {
        Log.d("Lifecycle", "FeedFragment onDestroy")
        super.onDestroy()
    }
}
```

Листинг 4 - ProfileFragment

```
class ProfileFragment : Fragment() {

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

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        Log.d("Lifecycle", "ProfileFragment onCreate")
    }

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

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

        binding.apply {
            nameText.text = "Glafira Smirnova"
            usernameText.text = "@glafira"
            emailText.text = "glafira@gmail.com"
            statusText.text = "Online"
            bioText.text = "Beginner Android dev📱"
        }
    }

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

Листинг 5 - SettingsFragment

```
class SettingsFragment : Fragment() {

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

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d("Lifecycle", "SettingsFragment onCreate")
    }

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

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

```
        super.onViewCreated(view, savedInstanceState)

        val prefs = requireContext().getSharedPreferences("settings",
Context.MODE_PRIVATE)
        val isDark = prefs.getBoolean("dark_theme", false)
        binding.themeSwitch.isChecked = isDark

        binding.themeSwitch.setOnCheckedChangeListener { _: CompoundButton,
checked: Boolean ->
            prefs.edit().putBoolean("dark_theme", checked).apply()
            AppCompatDelegate.setDefaultNightMode(
                if (checked) AppCompatDelegate.MODE_NIGHT_YES
                else AppCompatDelegate.MODE_NIGHT_NO
            )
        }
    }

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

На выходе было получено корректно запускаемое Android-приложение с базовой архитектурой, навигацией между экранами и заделом для дальнейшего расширения функциональности.

Лабораторная работа 2

Тема: Управление состоянием приложения и архитектура MVVM

Задачи:

1) внедрение архитектуры MVVM;
2) добавление ViewModel для хранения состояния профиля и настроек;
3) реализация редактируемых данных профиля (имя, статус);
4) добавление переключателя темы (светлая / тёмная);
5) сохранение состояния при повороте экрана;
6) логирование жизненного цикла ViewModel;
7) реактивное обновление UI через LiveData.

Был создан общий AppViewModel (см. Листинг 6), используемый несколькими экранами. В нём хранятся:

- имя пользователя;
- статус пользователя;
- состояние темы приложения.

Экран профиля отображает данные из ViewModel и позволяет редактировать их. Редактирование реализовано через режим «Редактировать → Сохранить» (см. Листинг 7), при котором изменения применяются только после нажатия кнопки сохранения (см. Риснок 3).

Экран настроек содержит переключатель темы, состояние которого хранится во ViewModel (см. Листинг 8). При изменении значения происходит переключение темы всего приложения (см. Риснок 4).

Для всех данных использованы LiveData, что обеспечивает автоматическое обновление интерфейса. При повороте экрана состояние данных сохраняется (см. Риснок 5).

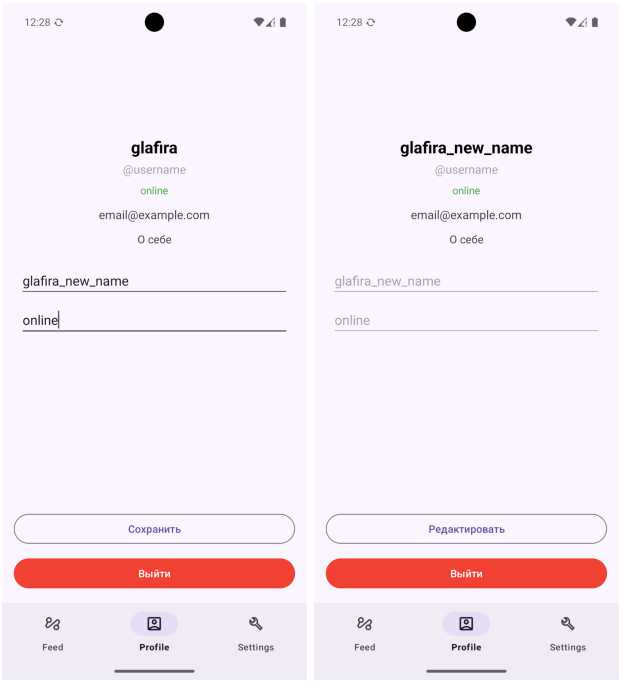

Рисунок 3 - Экран профиля в режиме просмотра и редактирования

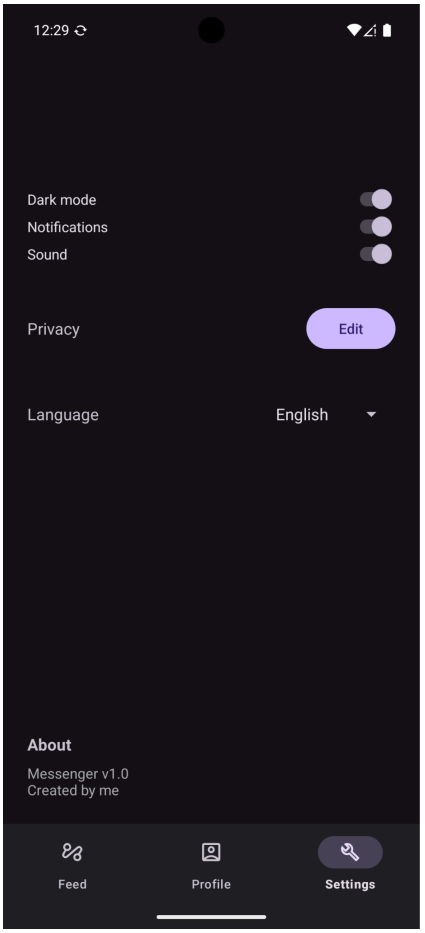

Рисунок 4 - Экран настроек с переключением темы

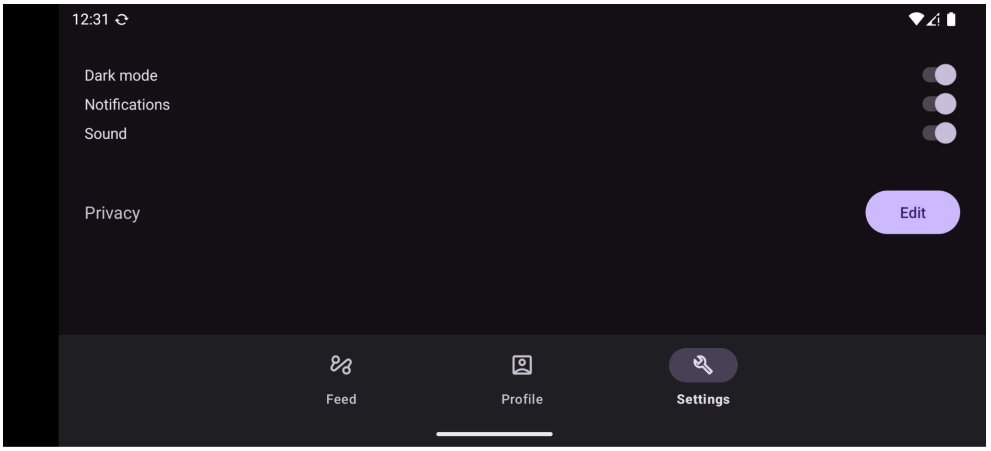

Рисунок 5 - Сохранение состояния при повороте экрана

Листинг 6 - AppViewModel.kt

```
class AppViewModel : ViewModel() {

    private val _name = MutableLiveData("glafira")
    val name: LiveData<String> = _name
```

```
    private val _status = MutableLiveData("online")
    val status: LiveData<String> = _status

    private val _isDark = MutableLiveData(false)
    val isDark: LiveData<Boolean> = _isDark

    init {
        Log.d("VM", "AppViewModel init")
    }

    fun updateName(newName: String) { _name.value = newName }
    fun updateStatus(newStatus: String) { _status.value = newStatus }
    fun setDarkMode(enabled: Boolean) { _isDark.value = enabled }

    override fun onCleared() {
        Log.d("VM", "AppViewModel onCleared()")
        super.onCleared()
    }
}
```

Листинг 7 - ProfileFragment.kt

```
class ProfileFragment : Fragment() {

    private var _binding: FragmentProfileBinding? = null
    private val binding get() = _binding!!
    private val appVM: AppViewModel by activityViewModels()

    private var isEditing = false


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d("Lifecycle", "ProfileFragment onCreate")
    }

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

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

        appVM.name.observe(viewLifecycleOwner) { binding.nameText.text = it }
        appVM.status.observe(viewLifecycleOwner) { binding.statusText.text =
it }
```

```
        appVM.name.observe(viewLifecycleOwner) { if (!isEditing)
binding.nameEdit.setText(it) }
        appVM.status.observe(viewLifecycleOwner) { if (!isEditing)
binding.statusEdit.setText(it) }

        binding.editProfileButton.setOnClickListener {
            if (!isEditing) {
                isEditing = true
                binding.editProfileButton.text = "Сохранить"
                binding.nameEdit.isEnabled = true
                binding.statusEdit.isEnabled = true
            } else {
                isEditing = false
                binding.editProfileButton.text = "Редактировать"
                binding.nameEdit.isEnabled = false
                binding.statusEdit.isEnabled = false

                appVM.updateName(binding.nameEdit.text.toString())
                appVM.updateStatus(binding.statusEdit.text.toString())
            }
        }
    }

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

Листинг 8 - SettingsFragment.kt

```
class SettingsFragment : Fragment() {

    private var _binding: FragmentSettingsBinding? = null
    private val binding get() = _binding!!
    private val appVM: AppViewModel by activityViewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d("Lifecycle", "SettingsFragment onCreate")
    }

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

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

        appVM.isDark.observe(viewLifecycleOwner) { isDark ->
            if (binding.themeSwitch.isChecked != isDark) {
                binding.themeSwitch.isChecked = isDark
            }
        }

        binding.themeSwitch.setOnCheckedChangeListener { _, checked ->
            appVM.setDarkMode(checked)
        }
    }

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

В результате была реализована архитектура MVVM, обеспечивающая разделение логики и интерфейса, сохранение состояния приложения и реактивное обновление UI.

Лабораторная работа 3

Тема: Загрузка данных из API и офлайн-доступ с использованием Room

Задачи:

1) загрузить список сообщений из публичного API;
2) реализовать сетевое взаимодействие через Retrofit и корутины;
3) отобразить данные в RecyclerView;
4) сохранить сообщения в локальную базу данных Room;
5) обеспечить загрузку данных из базы при отсутствии сети;
6) добавить кнопку ручного обновления;
7) реализовать Repository как единую точку доступа к данным.

Была реализована модель данных сообщений, Retrofit-клиент для загрузки данных и база данных Room с использованием Entity, Dao (см. Листинг 9) и Database.

Создан MessageRepository (см. Листинг 10), который:

- загружает данные из сети;

- сохраняет их в локальную базу;
- при отсутствии сети возвращает данные из базы.

На экране ленты данные отображаются с помощью RecyclerView. Кнопка «Обновить» инициирует повторную загрузку данных. Также отображается информация о времени последнего обновления.

Все сетевые операции выполняются в корутинах, а ViewModel отвечает за взаимодействие между UI и репозиторием (см. Листинг 11).

Листинг 9 - MessageDao.kt

```
@Dao
interface MessageDao {

    @Query("SELECT * FROM messages ORDER BY id DESC")
    suspend fun getAll(): List<MessageEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(items: List<MessageEntity>)

    @Query("DELETE FROM messages")
    suspend fun clear()
}
```

Листинг 10 - MessageRepository.kt

```
class MessageRepository(
    private val api: ApiService,
    private val dao: MessageDao
) {
    suspend fun loadMessages(): List<MessageEntity> {
        return try {
            val remote = api.getMessages()
            val entities = remote.map { it.toEntity() }
            dao.insertAll(entities)
            entities
        } catch (e: IOException) {
            // нет сети
            dao.getAll()
        } catch (e: Exception) {
            // сломался парсинг
            dao.getAll()
        }
    }

    suspend fun refresh(): List<MessageEntity> {
        return loadMessages()
```

```
    }
}
```

Листинг 11 - FeedViewModel.kt

```
class FeedViewModel(
    private val repo: MessageRepository
) : ViewModel() {

    private val _messages = MutableLiveData<List<MessageEntity>>(emptyList())
    val messages: LiveData<List<MessageEntity>> = _messages

    private val _lastUpdated = MutableLiveData("--")
    val lastUpdated: LiveData<String> = _lastUpdated

    private fun nowTime(): String =
        java.text.SimpleDateFormat("HH:mm:ss", java.util.Locale.getDefault())
            .format(java.util.Date())

    fun load() {
        viewModelScope.launch(Dispatchers.IO) {
            val data = repo.loadMessages()
            _messages.postValue(data)
            _lastUpdated.postValue(nowTime())
        }
    }

    // поскольку из апишки приходят статичные данные на любой запрос,
    // добавим поле "Последнее обновление" и докажем работоспособность
кнопки "Обновить")
    fun refresh() {
        viewModelScope.launch(Dispatchers.IO) {
            val data = repo.refresh()
            _messages.postValue(data)
            _lastUpdated.postValue(nowTime())
        }
    }
}
```

Было реализовано получение данных из сети с возможностью офлайн-доступа, что существенно повысило устойчивость и практическую ценность приложения.

Лабораторная работа 4

Тема: UI, интерактивность и фоновая работа

Задачи:

1) улучшение пользовательского интерфейса;

2) реализацию кастомного элемента списка сообщений (аватар, имя, текст);

3) применение компонентов Material Design;

4) добавление возможности «лайкать» сообщения;

5) настройку фоновой синхронизации через WorkManager;

6) показ уведомлений при успешной синхронизации;

7) запрос разрешений при запуске приложения;

8) корректную реакцию на события онлайн/офлайн;

9) соблюдение принципов MVVM.

Для сообщений был создан кастомный layout item_message.xml (см. Листинг 12) на основе MaterialCardView, включающий аватар, текст и кнопку лайка (см. Рисунок 6). Состояние лайка сохраняется в базе данных и не теряется при обновлении данных (см. Листинг 13).

Для интерфейса использованы компоненты Material Design: AppBarLayout, MaterialToolbar, FloatingActionButton.

Фоновая синхронизация реализована с помощью WorkManager, который периодически обновляет данные (см. Листинг 14). При успешном обновлении отображается системное уведомление (см. Рисунок 7, Листинг 15). Для Android 13+ реализован запрос разрешения на отправку уведомлений (см. Рисунок 8).

Также реализовано отслеживание состояния сети с помощью ConnectivityManager. При смене онлайн/оффлайн состояния пользователю отображается уведомление через Snackbar (см. Рисунок 9).

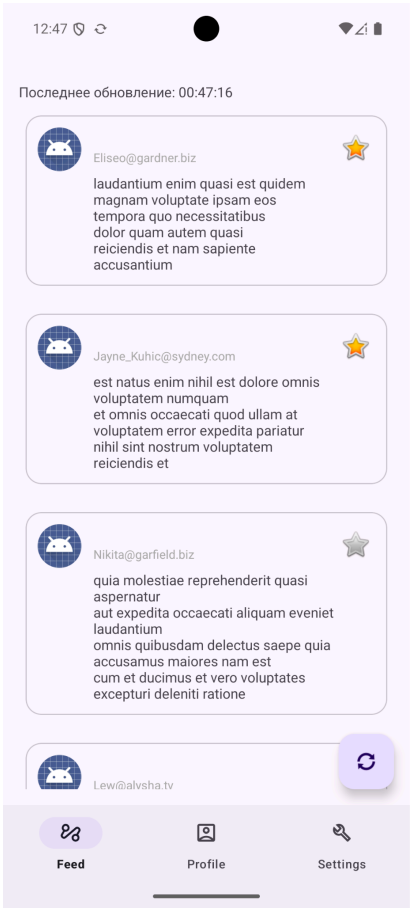

Рисунок 6 - Экран ленты с карточками сообщений

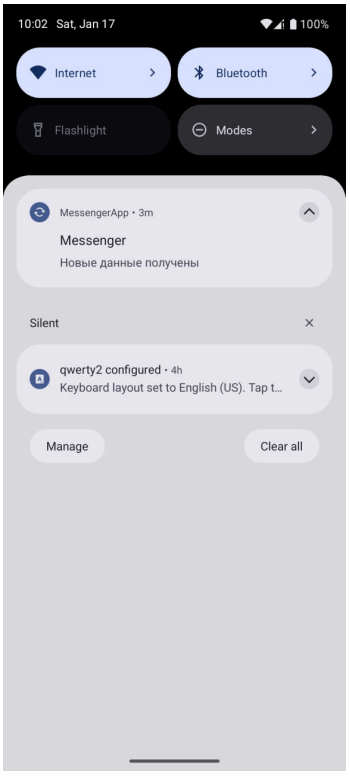

Рисунок 7 - Системное уведомление

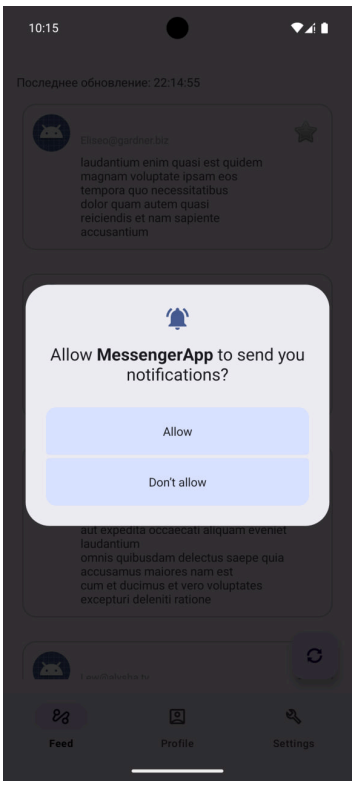

Рисунок 8 - Запрос приложением разрешений

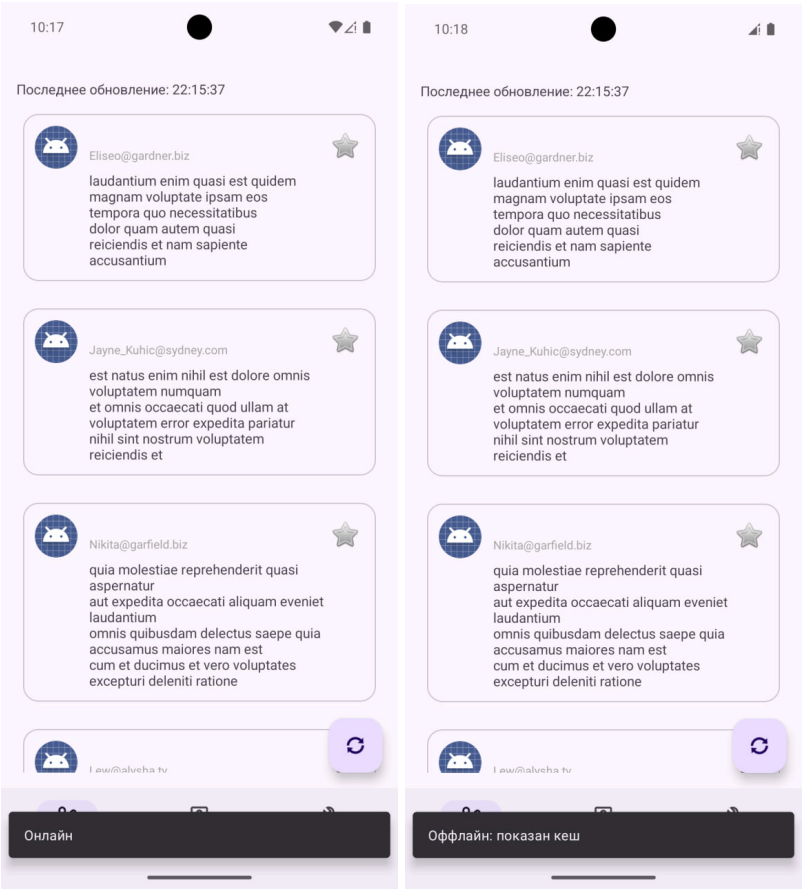

Рисунок 9 - Snackbar при включении/отключении интернета

Листинг 12 - Код item_message.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_marginBottom="12dp"
   app:cardCornerRadius="16dp"
   app:cardUseCompatPadding="true">

   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="horizontal"
       android:padding="12dp">

       <!-- аватар -->
       <ImageView
           android:id="@+id/avatarImage"
           android:layout_width="44dp"
           android:layout_height="44dp"
           android:layout_marginEnd="12dp"
           android:contentDescription="Avatar"
           android:src="@android:drawable/sym_def_app_icon" />

       <!-- текст -->
       <LinearLayout
           android:layout_width="0dp"
           android:layout_height="wrap_content"
           android:layout_weight="1"
           android:orientation="vertical">

           <TextView
               android:id="@+id/title"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:textStyle="bold"
               android:textSize="15sp" />

           <TextView
               android:id="@+id/author"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:layout_marginTop="2dp"
               android:textColor="@android:color/darker_gray"
               android:textSize="12sp" />

           <TextView
               android:id="@+id/body"
```

```
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:textSize="14sp" />
        </LinearLayout>

        <!-- лайк -->
        <ImageButton
            android:id="@+id/likeButton"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:background="@android:color/transparent"
            android:contentDescription="Like"
            android:src="@android:drawable/btn_star_big_off" />

    </LinearLayout>

</com.google.android.material.card.MaterialCardView>
```

Листинг 13 - MessageAdapter.kt

```
class MessageAdapter(
    private val onLikeClick: (Int) -> Unit
) : RecyclerView.Adapter<MessageAdapter.VH>() {

    private val items = mutableListOf<MessageEntity>()

    fun submitList(newItems: List<MessageEntity>) {
        items.clear()
        items.addAll(newItems)
        notifyDataSetChanged()
    }

    class VH(val b: ItemMessageBinding) : RecyclerView.ViewHolder(b.root)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): VH {
        val b =
ItemMessageBinding.inflate(LayoutInflater.from(parent.context), parent,
false)
        return VH(b)
    }

    override fun onBindViewHolder(holder: VH, position: Int) {
        val item = items[position]

        holder.b.author.text = item.author
        holder.b.body.text = item.text

        holder.b.likeButton.setImageResource(
            if (item.isLiked) android.R.drawable.btn_star_big_on
            else android.R.drawable.btn_star_big_off
```

```
        )

        holder.b.likeButton.setOnClickListener {
            onLikeClick(item.id)
        }
    }

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

Листинг 14 - SyncWorker.kt

```
class SyncWorker(appContext: Context, params: WorkerParameters) :
CoroutineWorker(appContext, params) {

    override suspend fun doWork(): Result {
        val db = Room.databaseBuilder(applicationContext,
AppDatabase::class.java, "messenger.db").build()
        val repo = MessageRepository(RetrofitClient.api, db.messageDao())

        return try {
            repo.refresh() // обновили кеш
            NotificationHelper.show(applicationContext, "Новые данные
получены")
            Result.success()
        } catch (e: Exception) {
            Result.retry()
        }
    }
}
```

Листинг 15 - NotificationHelper.kt

```
object NotificationHelper {
    private const val CHANNEL_ID = "sync_channel"

    fun show(ctx: Context, text: String) {
        val nm = ctx.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val ch = NotificationChannel(
                CHANNEL_ID, "Sync", NotificationManager.IMPORTANCE_DEFAULT
            )
            nm.createNotificationChannel(ch)
        }

        val notif = NotificationCompat.Builder(ctx, CHANNEL_ID)
            .setSmallIcon(android.R.drawable.stat_notify_sync)
            .setContentTitle("Messenger")
            .setContentText(text)
```

```
            .setAutoCancel(true)
            .build()

        nm.notify(1, notif)
    }
}
```

В результате была реализована полноценная интерактивная версия приложения мессенджера с современным интерфейсом, фоновыми задачами, уведомлениями и корректной обработкой системных событий.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано Android-приложение «Мессенджер», реализованное в соответствии с современными требованиями к архитектуре, пользовательскому интерфейсу и работе с данными. Разработка велась поэтапно в рамках четырёх лабораторных работ, что позволило последовательно усложнять функциональность приложения и закреплять полученные теоретические знания на практике.

В рамках работы были реализованы навигация между экранами, архитектура MVVM с использованием ViewModel и LiveData, загрузка данных из сетевого источника с сохранением в локальной базе данных Room, а также поддержка офлайн-режима. Приложение было дополнено интерактивными элементами интерфейса, выполненными с использованием компонентов Material Design, возможностью взаимодействия с элементами списка сообщений и фоновой синхронизацией данных при помощи WorkManager.

В результате поставленная цель курсовой работы была достигнута: создано работоспособное мобильное приложение, демонстрирующее основные принципы разработки Android-приложений и служащее основой для дальнейшего развития, включая добавление аутентификации пользователей, обмена сообщениями в реальном времени и интеграции с серверной частью.