Analiza projekta Bluetooth chat and reminder app

Marija Marković
Verifikacija softvera

22. oktobar 2025.

1 Opis projekta

Bluetooth chat and reminder app je Android aplikacija razvijena u okviru studentskog projekta na kursu Programske paradigme, na Matematičkom fakultetu, Univerziteta u Beogradu. Izvorni kod aplikacije možete pogledati ovde.
Aplikacija je namenjena za dopisivanje korišćenjem Bluetootha, a pored toga korisnici imaju mogućnost da dodaju i podsetnike za svoje dnevne aktivnosti. Za razvoj je korišćen programski jezik Kotlin u okviru okruženja Android Studio.

2 Cilj analize projekta

Cilj analize ovog projekta je da se primenom različitih alata za verifikaciju i testiranje utvrde slabosti sistema, potencijalni bagovi i mogućnosti za unapređenje. Fokus je na proveri stabilnosti i pouzdanosti koda kroz statičku i dinamičku analizu, čime se dobija uvid u ponašanje aplikacije u različitim uslovima.

3 Pronađeni defekti i ideje za poboljšanje aplikacije

1. Analizom postojećeg projekta može se uočiti da aplikacija nema jasno definisanu arhitekturu. Iako se u njenoj strukturi mogu prepoznati elementi osnovnog MVC (Model-View-Controller) pristupa, granice između pojedinačnih slojeva nisu precizno definisane. Zbog toga se poslovna logika često meša sa logikom korisničkog interfejsa unutar Activity klasa (kod za prikazivanje UI elementa meša se sa kodom za validaciju unosa korisnika, proces autentikacije, itd.). Takav pristup otežava razdvajanje odgovornosti, čini kod teže čitljivim i komplikuje njegovo dugoročno održavanje.

Predlog rešenja za arhitekturu projekta - korišćenje MVVM arhitekture:

```
// Data layer - Repository
interface ChatRepository {
    suspend fun sendMessage(message: ChatMessage): Result<Unit>
    suspend fun connectToDevice(address: String): Result<Unit>
    fun observeMessages(): Flow<List<ChatMessage>>
    fun observeConnectionState(): Flow<ConnectionState>
}

class ChatRepositoryImpl(
    private val bluetoothService: BluetoothChatService,
    private val localDataSource: ChatLocalDataSource
) : ChatRepository {

    // implementation
}

// Presentation layer - ViewModel
class ChatViewModel(
    private val chatRepository: ChatRepository
) : ViewModel() {
    ...
    fun sendMessage(content: String) {
        viewModelScope.launch {
            _uiState.value = ChatUiState.Loading

            val message = ChatMessage(
                content = content,
                timestamp = System.currentTimeMillis(),
                isFromUser = true
            )

            val result = chatRepository.sendMessage(message)
            _uiState.value = when {
                result.isSuccess -> ChatUiState.Success
                else -> ChatUiState.Error(result.exceptionOrNull
                    ()?.message)
            }
        }
    }
    ...
}

// UI layer - Activity (samo UI logika)
class ChatActivity : AppCompatActivity() {
    private lateinit var viewModel: ChatViewModel

```

```
    ...

    private fun setupClickListeners() {
        binding.sendBtn.setOnClickListener {
            val message = binding.enterMessage.text.toString()
            if (message.isNotBlank()) {
                viewModel.sendMessage(message)
                binding.enterMessage.text.clear()
            }
        }
    }
    ...
}
```

2. Projekat koristi Firebase za autentikaciju, ali, umesto da je logika centralizovana i dostupna svim delovima aplikacije, različiti `Activity` fajlovi imaju svoje načine rukovanja autentikacijom. Zbog ovoga je testiranje logike koja uključuje autentikaciju praktično onemogućeno. Predlog je da se trenutni pristup zameni korišćenjem *Dependency Injection*, npr. kroz Hilt, kako bi se logika centralizovala i olakšalo testiranje.

   Predlog:

```
// FirebaseModule.kt
@Module
@InstallIn(SingletonComponent::class)
object FirebaseModule {
    @Provides
    @Singleton
    fun provideFirebaseAuth(): FirebaseAuth = FirebaseAuth.
       getInstance()
}

// AuthManager.kt
class AuthManager @Inject constructor(private val auth:
   FirebaseAuth) {
    fun signIn(email: String, pass: String, cb: (Boolean) ->
       Unit) {
        auth.signInWithEmailAndPassword(email, pass)
            .addOnCompleteListener { cb(it.isSuccessful) }
    }
}

// LoginActivity.kt
@AndroidEntryPoint
class LoginActivity : AppCompatActivity() {
    @Inject lateinit var authManager: AuthManager

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

```
24          super.onCreate(savedInstanceState)
25          setContentView(R.layout.activity_login)
26
27          authManager.signIn("email@example.com", "password") {
               success ->
28              Toast.makeText(this, if (success) "Uspesno!" else "
                   Greska", Toast.LENGTH_SHORT).show()
29          }
30      }
31  }
```

3. Za automatsku statičku analizu kvaliteta koda korišćen je alat **Detekt**. Detekt je statički analizator koda za Kotlin koji automatski identifikuje probleme sa kvalitetom koda, kompleksnošću, čitljivošću, itd. Analiza je pokazala da aplikacija ima značajno prisustvo nekonzistentnosti u stilu pisanja koda i poštovanju konvencija.
   Ukupni rezultati:
   Identifikovano je 70 problema u kodu. Najproblematičniji fajlovi: `ChatActivity.kt`, `BluetoothChatService.kt`, `DeviceListActivity.kt`.
   Glavne kategorije problema: stilovi pisanja koda, naming konvencije, hard-kodiranje konstanti. Dodatno, kod ima značajan broj `TODO` komentara, kao i samo zakomentarisanog koda koji se ne koristi. Pronađeni su slučajevi generičkog upravljanja izuzecima i neiskorišćenih delova koda.

4. **KtLint** je Kotlin linter i formater koji proverava poštovanje Kotlin coding standarda i konvencija. Pokretanjem ovog alata kod je automatski formatiran i stil koda je ujednačen. Za razliku od problema koje pronalazi Detekt, koji uglavnom ukazuju na potencijalne greške ili lošu praksu, problemi koje detektuje KtLint su više stilske prirode. Ipak, nepoštovanje stilskih konvencija može značajno otežati rad većih timova na istom softveru.

5. Aplikacija sadrži nepotpune TODO komentare umesto implementacije upravljanja dozvolama, što može dovesti do pada aplikacije kada korisnik pokuša da koristi Bluetooth bez potrebnih dozvola. Potrebno je implementirati odgovarajući tok zahteva za dozvole sa opcijama za slučaj kada korisnik odbije dozvolu.

```
if (ActivityCompat.checkSelfPermission(
        this,
        android.Manifest.permission.BLUETOOTH_SCAN
    ) != PackageManager.PERMISSION_GRANTED
) {
    // TODO: Consider calling
    //    ActivityCompat#requestPermissions
    // here to request the missing permissions, and then overriding
    //   public void onRequestPermissionsResult(int requestCode, String[] permissions,
    //                                          int[] grantResults)
    // to handle the case where the user grants the permission. See the documentation
    // for ActivityCompat#requestPermissions for more details.

}
```

6. Validacija unosa: Nedostaje provera korisničkih unosa, što može dovesti do bezbednosnih problema, ili do slanja predugačkih poruka koje mogu izazvati pad aplikacije. Potrebno je dodati proveru dužine poruka, čišćenje nepoželjnih karaktera i ograničiti dozvoljene tipove podataka.

7. Curenje memorije: Handler objekti u ChatActivity čuvaju reference na Activity, što može izazvati curenje memorije kada se Activity zatvori. Potrebno je koristiti WeakReference ili pravilno očistiti Handler u onDestroy metodi.
U ovom slučaju, Handler je privatna klasa unutar ChatActivity klase i ima "jaku"referencu na nju. Ako se u međuvremenu Activity zatvori, ali Handler i dalje ima poruke u redu, onda će ChatActivity ostati u memoriji i neće moći da se oslobodi.

```
open class ChatActivity : AppCompatActivity() {
        strings.clear()
        myarrayAdapter.notifyDataSetChanged()
    }
    private val mHandler = @SuppressLint("HandlerLeak")
    object : Handler() {
        override fun handleMessage(msg: Message) {
            when (msg.what) {
```

Predlog rešenja:

```
private class ChatHandler(activity: ChatActivity) : Handler() {
    private val activityRef = WeakReference(activity)

    override fun handleMessage(msg: Message) {
        val activity = activityRef.get() ?: return  // Activity
            doesn't exist anymore, return
```

8. NotificationActivity koristi statičke nizove za upravljanje alarmima, što može izazvati curenje memorije jer statičke reference sprečavaju oslobađanje memorije. Rešenje je korišćenje obrasca Repository uz pravilno upravljanje memorijom. Predlog rešenja:

```
class NotificationActivity : AppCompatActivity() {

    companion object{
        var minutes: List<Int> = ArrayList(25)
        var alarmManagers = arrayOfNulls<AlarmManager>(25)
        var intents = arrayOfNulls<PendingIntent>(25)
        var info = 0

    }
```

Umesto statičkih nizova, može se koristiti AlarmRepository koji će upravljati alarmima, primer:

```
object AlarmRepository {
    private val alarms = mutableListOf<AlarmData>()

    data class AlarmData(
        val id: Int,
        val timeInMillis: Long,
        val message: String
    )
    fun addAlarm(alarm: AlarmData) {
        alarms.add(alarm)
    }
    fun removeAlarm(id: Int) {
        alarms.removeAll { it.id == id }
    }
    fun getAll(): List<AlarmData> = alarms.toList()
}
```

A zatim u NotificationActivity samo koristimo:

```
AlarmRepository.addAlarm(
    AlarmRepository.AlarmData(
        notificationId,
        alarmStart,
        notification
    )
)
```

9. Još problema sa testiranjem: Bluetooth funkcionalnosti ne mogu biti testirane jer su direktno vezane za Android framework. Potrebno je kreirati wrapper interface-e koji mogu biti mock-ovani za testiranje, što zahteva izmenu izvornog koda. Primer:

```
    interface BluetoothConnector {
    fun connectToDevice(address: String): Boolean
    fun sendMessage(message: String)
    fun receiveMessage(): String?
    }
```

Produkcioni kod:

```
    class RealBluetoothConnector : BluetoothConnector {
    private val adapter = BluetoothAdapter.getDefaultAdapter()

    override fun connectToDevice(address: String): Boolean {
        val device = adapter.getRemoteDevice(address)
        val socket = device.createRfcommSocketToServiceRecord(
           MY_UUID)
        socket.connect()
        return true
    }
```

```
10
11      override fun sendMessage(message: String) {
12          // implementation
13      }
14
15      override fun receiveMessage(): String? {
16          // implementation
17      }
18  }
```

U testu:

```
1       class FakeBluetoothConnector : BluetoothConnector {
2       val sentMessages = mutableListOf<String>()
3       override fun connectToDevice(address: String) = true
4       override fun sendMessage(message: String) { sentMessages.add
           (message) }
5       override fun receiveMessage() = "Hello test"
6   }
```

10. Nedostatak lenjog učitavanja: Sve poruke se učitavaju odjednom, što može dovesti do problema sa performansama kada ih ima mnogo. Potrebno je implementirati neki vid keširanja ili postepenog učitavanja. Primer biblioteka: RecyclerView, Paging.

11. Nema čuvanja podataka lokalno: Poruke se čuvaju samo u memoriji i gube kada se aplikacija zatvori. To loše utiče na korisničko iskustvo, jer korisnici gube istoriju razgovora.

4 Zaključak

Analiza projekta je pokazala da aplikacija ima značajne nedostatke u arhitekturi, kvalitetu koda i rukovanju resursima. Nedostatak centralizovane logike, loša separacija slojeva, nepotpuna validacija unosa, problemi sa curenjem memorije i nedostatak lokalnog čuvanja podataka ukazuju na potencijalne rizike po stabilnost i pouzdanost aplikacije.

Poseban problem predstavlja testiranje: zbog čvrste povezanosti logike sa Android frameworkom, neadekvatnog integrisanja autentikacije i direktnog korišćenja Bluetooth funkcionalnosti, praktično je nemoguće automatizovano testirati ključne delove aplikacije. Da bi testiranje postalo izvodljivo, potrebno je uvesti jasnu arhitekturu, dependency injection, interfejse za testabilne komponente i mehanizme za izolaciju resursa.

Sve u svemu, trenutna implementacija ozbiljno otežava održavanje, proširivanje i proveru ispravnosti aplikacije, te zahteva značajne izmene kako bi se omogućilo stabilno i pouzdano korišćenje, kao i efikasno testiranje.