introduzione.kt

workshop.css

‘oneminute’ {

[Architettura e Implementazione]

< Integrazione AI per Riassunti Automatici e Gestione di Notizie in Tempo Reale >

}

Luca Timpano 240236

Greta Tigano 239610

Breve introduzione {oneminute}

L'applicazione descritta è una piattaforma per la visualizzazione di notizie che utilizza un'architettura MVVM (Model-View-ViewModel) e sfrutta tecnologie come Room per il database locale, Retrofit per le chiamate API e LiveData/ViewModel per la gestione dello stato dei dati.

Quest'applicazione è costituita da:

1. UI
2. Repository
3. API
4. Database Room
5. Models

}

Parte UI{MainActivity_news}

Questa classe rappresenta l'attività principale dell'applicazione ed è responsabile della coordinazione tra i fragment e la navigazione.

1. **Inizializzazione del ViewModel**: Viene creato un NewsViewModel e un NewsViewModeNearMe per gestire i dati relativi alle notizie e alla posizione geografica.

2. **Navigazione:** Utilizza un NavController per gestire la navigazione tra i fragment.

3. **Bottom Navigation:** Il BottomNavigationView consente di spostarsi tra le sezioni principali (Home, Search, Favourites, Near Me).

}

MainActivity{Logica principale}

```
class MainActivity_news : AppCompatActivity() {
    lateinit var newsViewModel: NewsViewModel
    lateinit var newsViewModeNearMe: NewsViewModeNearMe

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

        // Inizializzazione del ViewModel
        val newsRepository = NewsRepository(ArticleDatabase(this))
        val viewModelProviderFactory = NewsViewModelProviderFactory(application, newsRepository)
        newsViewModel = ViewModelProvider(this, viewModelProviderFactory).get(NewsViewModel::class.java)
        newsViewModeNearMe = ViewModelProvider(this,
viewModelProviderFactory).get(NewsViewModeNearMe::class.java)

        // Setup del NavController
        val navHostFragment = supportFragmentManager.findFragmentById(R.id.newsNavHostFragment) as
NavHostFragment
        val navController = navHostFragment.navController
        binding.bottomNavigationView.setupWithNavController(navController)
    }
}
```

}

Parte UI{HomePage Fragment}

Questo fragment visualizza le notizie principali (headlines) utilizzando paginazione automatica.

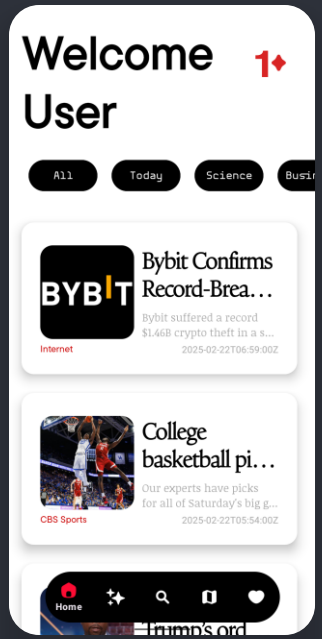
1. **Chiamata API:** Recupera le notizie principali dal paese selezionato (default “us").
2. **Paginazione:** Quando l'utente scorre verso il basso, vengono caricate altre notizie.
3. **Gestione Errori:** Mostra messaggi di errore in caso di problemi di connessione o assenza di risultati.

}

HomePage Fragment {Logica Principale}

```
}
```

```
class HomePage_fragment : Fragment(R.layout.fragment_home_page_fragment) {
    lateinit var newsViewModel: NewsViewModel
    lateinit var newsAdapter: NewsAdapter

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

        // Inizializzazione del ViewModel
        newsViewModel = (activity as MainActivity_news).newsViewModel

        // Setup RecyclerView
        setupHeadLinesRecycler()

        // Osservazione dei dati
        newsViewModel.headlines.observe(viewLifecycleOwner, Observer { response ->
            when (response) {
                is Resource.Success -> {
                    hideProgressBar()
                    response.data?.let { newsResponse ->
                        newsAdapter.differ.submitList(newsResponse.articles.toList())
                    }
                }
                is Resource.Error -> {
                    hideProgressBar()
                    Toast.makeText(activity, "Error: ${response.message}",
Toast.LENGTH_LONG).show()
                }
                is Resource.Loading -> {
                    showProgressBar()
                }
            }
        })
    }
}
```



Parte UI{AIchatfragment}

Questo fragment implementa una chatbot integrata con un modello generativo (Gemini).

1. **Interfaccia Utente:** L'utente inserisce una domanda e riceve una risposta generata dal modello.

2. **Richiesta API:** È stato usato l'SDK di Google *google.ai.client.generativeai*, che semplifica l'interazione con i modelli generativi come Gemini, gestendo automaticamente le chiamate HTTP e fornendo metodi pronti per generare contenuti.

}

AIchat{Logica principale}

```
class AIchatfragment : Fragment() {
    private val generativeModel = GenerativeModel(
        modelName = "gemini-1.5-flash",
        apiKey = BuildConfig.API_KEY
    )

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState:
Bundle?): View? {
        binding.promptButton.setOnClickListener {
            val inputText = binding.inputTextField.text.toString()
            if (inputText.isNotEmpty()) {
                generateResponse(initialPrompt + inputText)
            } else {
                Toast.makeText(requireContext(), "Please enter a question", Toast.LENGTH_SHORT).show()
            }
        }
        return binding.root
    }

    private fun generateResponse(inputText: String) {
        lifecycleScope.launch {
            try {
                val response = withContext(Dispatchers.IO) {
                    generativeModel.generateContent(inputText)
                }
                binding.outputTextView.text = response.text
            } catch (e: Exception) {
                Toast.makeText(requireContext(), "Error generating response", Toast.LENGTH_SHORT).show()
            } finally {
                binding.progressBar.visibility = View.GONE
            }
        }
    }
}
```

Parte UI{Search Fragment}

Questo fragment permette all'utente di cercare notizie basandosi su una query di ricerca.

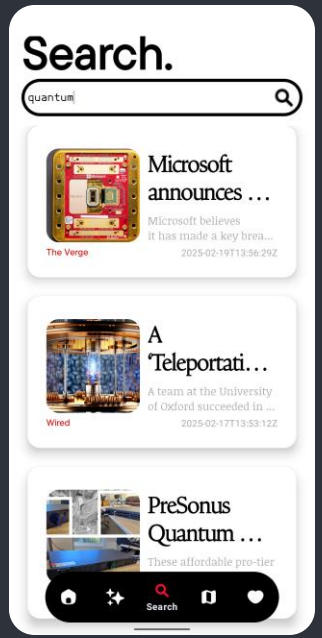
1. **Debounce Input:** La ricerca viene eseguita solo dopo che l'utente smette di digitare per 500ms.
2. **Paginazione:** Come nel fragment precedente, le ricerche supportano la paginazione.
3. **Gestione Errori**: Mostra messaggi di errore in caso di problemi di connessione.

}

Search Fragment{Logica Principale}

```
}
```

```
class search_fragment : Fragment(R.layout.fragment_search_fragment) {
    lateinit var newsViewModel: NewsViewModel
    lateinit var newsAdapter: NewsAdapter

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

        // Inizializzazione del ViewModel
        newsViewModel = (activity as MainActivity_news).newsViewModel

        // Setup RecyclerView
        setupSearchRecycler()

        // Listener per la barra di ricerca
        var job: Job? = null
        binding.searchEdit.addTextChangedListener { editable ->
            job?.cancel()
            job = MainScope().launch {
                delay(Constants.SEARCH_NEWS_TIME_DELAY)
                editable?.let {
                    if (editable.toString().isNotEmpty()) {
                        newsViewModel.searchNews(editable.toString())
                    }
                }
            }
        }
    }
}
```



Parte UI{Favourite Fragment}

Questo fragment visualizza gli articoli salvati come preferiti dall'utente.

1. **Swipe-to-Delete:** L'utente può eliminare un articolo preferito scivolando a sinistra o destra.

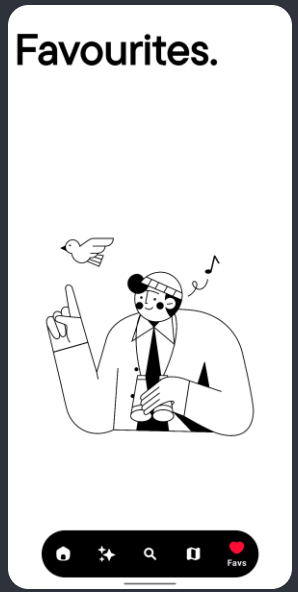
2. **Empty State:** Se non ci sono preferiti, viene mostrata un'immagine di placeholder.

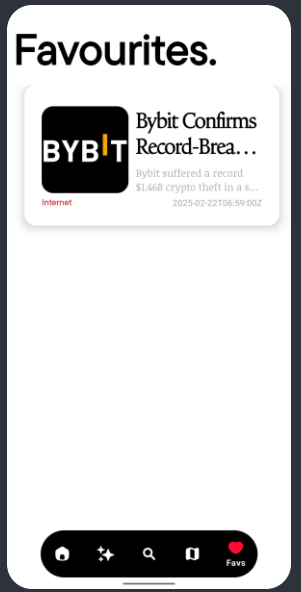
}

Favourite Fragment{Logica Principale}

```
}
```

```
class Favourite_fragment : Fragment(R.layout.fragment_favourite_fragment) {
    lateinit var newsViewModel: NewsViewModel
    lateinit var newsAdapter: NewsAdapter

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

        // Inizializzazione del ViewModel
        newsViewModel = (activity as MainActivity_news).newsViewModel

        // Setup RecyclerView
        setupFavouriteRecycler()

        // Swipe-to-Delete
        val itemTouchHelperCallback = object : ItemTouchHelper.SimpleCallback(
            ItemTouchHelper.UP or ItemTouchHelper.DOWN,
            ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT
        ) {
            override fun onSwiped(viewHolder: ViewHolder, direction: Int) {
                val position = viewHolder.adapterPosition
                val article = newsAdapter.differ.currentList[position]
                newsViewModel.deleteArticle(article)
                Snackbar.make(view, "Removed from favourites", Snackbar.LENGTH_LONG).apply {
                    setAction("Undo") {
                        newsViewModel.addToFavourites(article)
                    }
                    show()
                }
            }
        }
        ItemTouchHelper(itemTouchHelperCallback).attachToRecyclerView(binding.recyclerFavourites)
    }
}
```





Parte UI{Near Me}

Questo fragment mostra notizie locali basate sulla posizione dell'utente.

1. **Localizzazione:** Utilizza il servizio GPS per determinare la città dell'utente.
2. **Chiamata API:** Cerca notizie relative alla città identificata.
3. **Retry Button:** Consente all'utente di riprovare nel caso cambiasse città o volesse aggiornare le notizie.

}

forbeginners.html

workshop.css

Near Me{Logica Principale}

```
}
```

```
class NearMe_fragment : Fragment(R.layout.fragment_near_me_fragment), LocationListener {
    lateinit var newsViewModel: NewsViewModeNearMe
    lateinit var newsAdapter: NewsAdapter

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

        // Inizializzazione del ViewModel
        newsViewModel = (activity as MainActivity_news).newsViewModeNearMe

        // Ottieni la posizione corrente
        checkLocationPermission()

        // Setup RecyclerView
        setupNearMeRecycler()

        // Listener per il cambio di città
        newsViewModel.cityAlreadyFound.observe(viewLifecycleOwner) { found ->
            if (found) {
                binding.roundButton.visibility = View.VISIBLE
            } else {
                Toast.makeText(requireContext(), "Fetching city...", Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun checkLocationPermission() {
        when {
            ContextCompat.checkSelfPermission(
                requireContext(),
                android.Manifest.permission.ACCESS_FINE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED -> {
                getLocation()
            }
            else -> {
                permissionLauncher.launch(android.Manifest.permission.ACCESS_FINE_LOCATION)
            }
        }
    }
}
```

Programming Language

infographics.css

Parte UI{Article Fragment}

Questo fragment visualizza i dettagli di un articolo selezionato.

1. **Riassunto Automatico:** Utilizza un modello generativo (Gemini) per creare un riassunto della notizia.

2. **Aggiunta ai Preferiti:** L'utente può salvare l'articolo come preferito.

}

Article Fragment{Logica Principale}

```
class ArticleFragment : Fragment() {
    lateinit var newsViewModel: NewsViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState:
Bundle?): View {
        val view = inflater.inflate(R.layout.fragment_article, container, false)

        // Ottieni l'articolo passato tramite Safe Args
        val article: Article = args.article

        // Genera il riassunto
        generateResponse(initialPrompt + articleInfo)

        // Aggiungi all'elenco dei preferiti
        binding.roundButton.setOnClickListener {
            newsViewModel.addToFavourites(article)
            Toast.makeText(context, "Added to favourites", Toast.LENGTH_SHORT).show()
        }

        return view
    }
}
```

Repository{Funzione e Logica Principale}

Il repository funge da ponte tra il ViewModel e le fonti di dati (API e database locale). Gestisce la logica per recuperare, salvare ed eliminare dati. Il repository delega le **chiamate API** a Retrofit e le operazioni sul database al DAO.

```
class NewsRepository(val db: ArticleDatabase) {
    suspend fun getHeadlines(countryCode: String, pageNumber: Int) =
        RetrofitInstance.api.getHeadlines(countryCode, pageNumber)

    suspend fun searchNews(searchQuery: String, pageNumber: Int) =
        RetrofitInstance.api.searchForNews(searchQuery, pageNumber)

    suspend fun upsert(article: Article) = db.getArticleDao().upsert(article)

    fun getFavouriteNews() = db.getArticleDao().getAlleArticles()

    suspend fun deleteArticle(article: Article) = db.getArticleDao().deleteArticle(article)
}
```

}

API { Funzione e Logica Principale }

L'interfaccia NewsAPI definisce i metodi per interagire con l'API esterna utilizzando Retrofit.

- I metodi **getHeadlines** e **searchForNews** consentono di recuperare notizie principali e cercare notizie basandosi su una query.

- L'utilizzo di **Retrofit** semplifica la gestione delle chiamate API.

}

```
interface NewsAPI {
    @GET("v2/top-headlines")
    suspend fun getHeadlines(
        @Query("country") countryCode: String = "it",
        @Query("page") pageNumber: Int = 1,
        @Query("apiKey") apiKey: String = Constants.API_KEY
    ): Response<NewsResponse>

    @GET("v2/everything")
    suspend fun searchForNews(
        @Query("q") searchQuery: String,
        @Query("page") pageNumber: Int = 1,
        @Query("apiKey") apiKey: String = Constants.API_KEY
    ): Response<NewsResponse>
}
```

Database Room{Funzione e Logica Principale}

Il database Room viene utilizzato per archiviare gli articoli preferiti localmente.

- Il database utilizza Room per gestire lo schema e le migrazioni.
- I convertitori (Converters) permettono di serializzare/deserializzare oggetti complessi come Source.

}

```
@Database(entities = [Article::class], version = 2)
@TypeConverters(Converters::class)
abstract class ArticleDatabase : RoomDatabase() {
    abstract fun getArticleDao(): ArticleDAO

    companion object {
        val MIGRATION_1_2 = object : Migration(1, 2) {
            override fun migrate(database: SupportSQLiteDatabase) {
                database.execSQL("ALTER TABLE articles RENAME TO articles_temp")
                // Aggiorna lo schema
            }
        }
    }
}
```

```
@Dao
interface ArticleDAO {

    //suspend significa che, la funzione, può essere eseguita in una coroutine per non bloccare il
thread principale.
    @Insert(onConflict =OnConflictStrategy.REPLACE )
    suspend fun upsert(article: Article): Long

    //esegue una query, prende tutto dalla tabella articles
    @Query("SELECT * FROM articles")
    fun getAlleArticles(): LiveData<List<Article>>


    //elimina un oggetto specifico della tabella
    @Delete
    suspend fun deleteArticle(article: Article)
}
```

Models{Funzione e Logica Principale}

I modelli rappresentano le entità utilizzate nell'applicazione.

- La classe **Article** rappresenta un articolo di notizia con campi come autore, titolo, descrizione, fonte, ecc.
- L'uso di **Room** facilita la persistenza dei dati locali.

}

```
data class Article(
    @PrimaryKey(autoGenerate = true) var id: Int? = null,
    val author: String = "Unknown Author",
    val content: String = "No Content",
    val description: String = "No Description",
    val publishedAt: String = "Unknown Date",
    var source: Source? = null,
    val title: String = "Untitled",
    val url: String = "No URL",
    val urlToImage: String = "No Image URL"
)
```