MARVEL App

By MEM

- Michele Tosi
- Anuar Elio Magliari
- Marina Sotiropoulos

Caratteristiche principali

- Lingua inglese
- Accesso ai dati da remoto
- Salvataggio dei personaggi sul dispositivo
- Meccanismo di ricerca personalizzata
- Ordinamento personaggi personalizzato

Schermata iniziale

- Breve SplashScreen all'avvio dell'App.
- Il Fragment principale si compone di:
  - Toolbar
  - Recycler View
  - Floating Action Button (FAB)
- La Toolbar comprende:
  - Search View
  - Options Menu
  - Navigation Menu
- La Recycler View consente la visualizzazione, tramite Card View, dei vari personaggi Marvel.
- Il FAB permette di passare al Fragment dedicato ai personaggi preferiti.

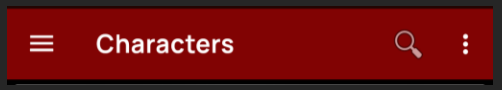
Splash Screen

Breve SplashScreen all'avvio dell'App.

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding:ActivityMainBinding

    private var characterDb: FavoriteCharactersDatabase?=null
    private var characterDao: FavoriteCharactersDao?=null

    companion object {
        private var SPLASHSCREEN: Long = 1500
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        supportActionBar?.hide()

        characterDb= FavoriteCharactersDatabase.getDatabase(this)
        characterDao=characterDb?.FavoriteCharactersDao()

        Handler(Looper.getMainLooper()).postDelayed({
            val intent = Intent(this, SecondActivity::class.java)
            startActivity(intent)
            finish()
        }, SPLASHSCREEN)

    }
}
```

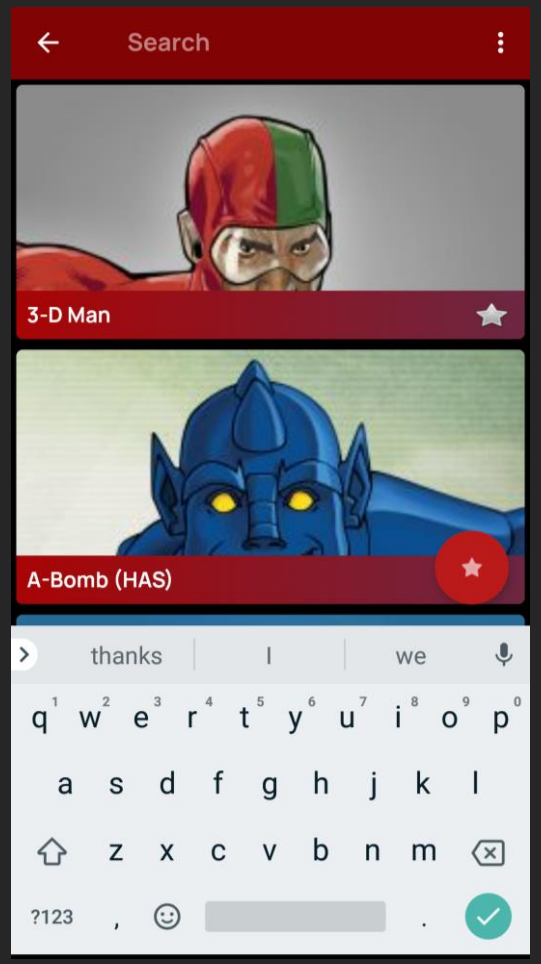
Search Bar

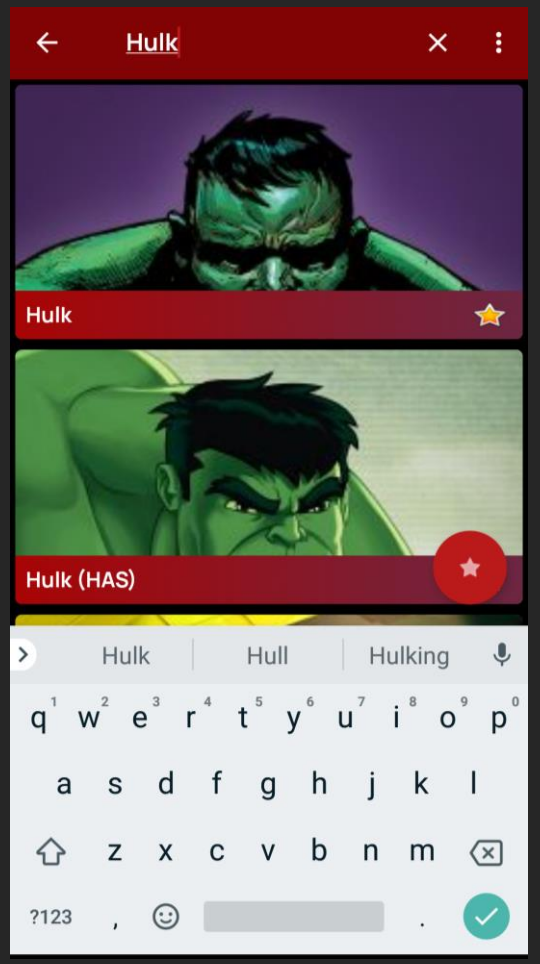
Cliccando sulla lente di ingrandimento della Toolbar si attiva la barra di ricerca.




```
<item
    android:id="@+id/menu_search"
    android:icon="@android:drawable/ic_menu_search"
    android:iconTint="@color/white"
    android:title="@string/menu_search"
    app:actionViewClass="androidx.appcompat.widget.SearchView"
    app:showAsAction="ifRoom|collapseActionView" />
```

```
    val searchItem =menu.findItem(R.id.menu_search)
    val searchView=searchItem.actionView as SearchView

    searchView.imeOptions=EditorInfo.IME_ACTION_DONE

    searchView.queryHint="Search"

    searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener{
        override fun onQueryTextSubmit(query: String?): Boolean {
            subscribeToList(query.toString(), order)
            name=query.toString()
            return true
        }

        override fun onQueryTextChange(newText: String?): Boolean {
            subscribeToList(newText.toString(), order)
            name=newText.toString()
            return true
        }

    })
}
```





Option Menu

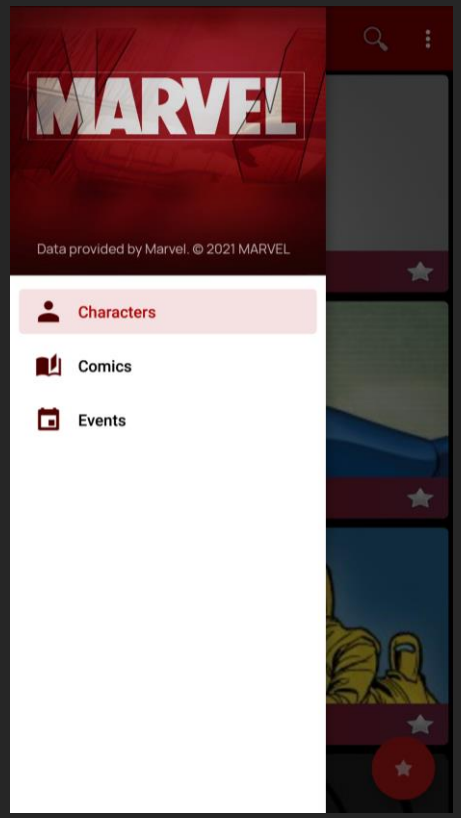
Cliccando su i tre puntini in alto a destra della Toolbar un menu a comparsa consente di scegliere il tipo di ordinamento desiderato per la visualizzazione dei personaggi.

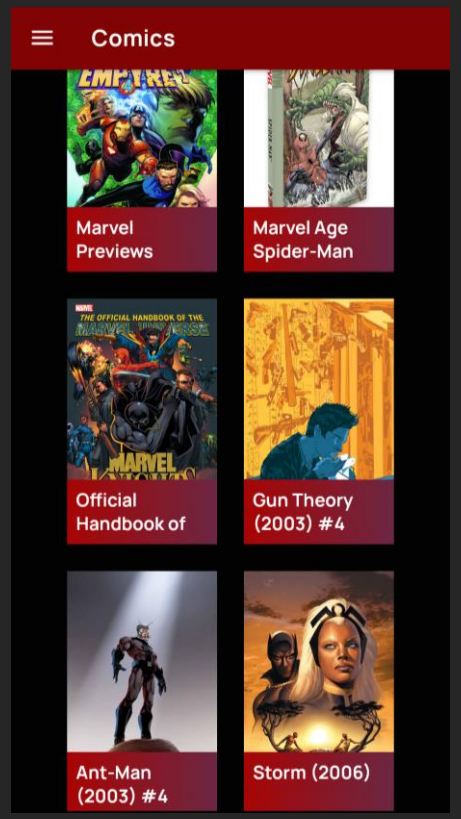

```
<item
    android:id="@+id/mnu_ordernameascending"
    android:iconTint="@android:color/darker_gray"
    android:title="@string/order_by_name_a_z" />

<item
    android:id="@+id/mnu_ordernamediscending"
    android:iconTint="@android:color/darker_gray"
    android:title="@string/order_by_name_z_a" />

<item
    android:id="@+id/mnu_ordermodifieddiscending"
    android:iconTint="@android:color/darker_gray"
    android:title="@string/order_by_recent_modified" />

<item
    android:id="@+id/mnu_ordermodifiedascending"
    android:iconTint="@android:color/darker_gray"
    android:title="@string/order_by_last_modified" />
```

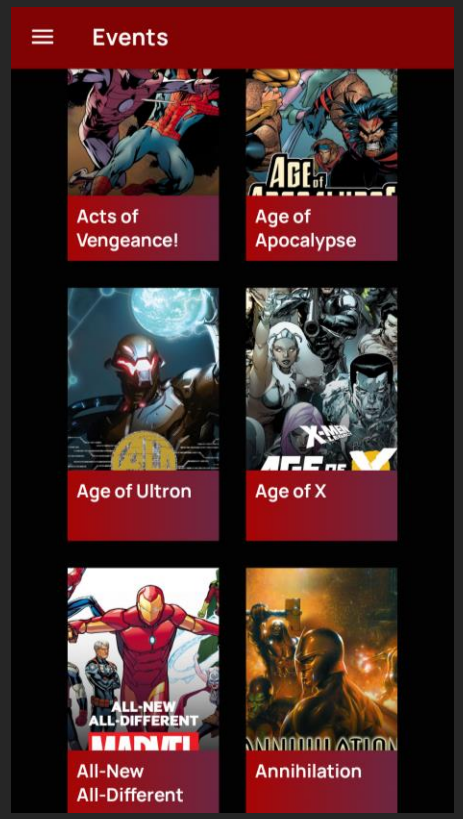
```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    order= when(item.itemId){
        R.id.mnu_ordernameascending ->  "name"
        R.id.mnu_ordernamediscending-> "-name"
        R.id.mnu_ordermodifiedascending-> "modified"
        R.id.mnu_ordermodifieddiscending-> "-modified"
        else -> order
    }

    subscribeToList(name, order)
    return super.onOptionsItemSelected(item)
}
```

Navigation Drawer Activity - 1

Cliccando sul menu in alto a sinistra della Toolbar si accede alla Navigation Drawer Activity che per permette all'utente di passare tra i diversi Fragmet disponibili: "Characters", "Comics" e "Events".







Navigation Drawer Activity - 2

```
class SecondActivity : AppCompatActivity() {

    private lateinit var appBarConfiguration: AppBarConfiguration

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)
        val toolbar: Toolbar = findViewById(R.id.toolbar)
        setSupportActionBar(toolbar)

        val drawerLayout: DrawerLayout = findViewById(R.id.drawer_layout)
        val navView: NavigationView = findViewById(R.id.nav_view)
        val navController = findNavController(R.id.nav_host_fragment)

        appBarConfiguration = AppBarConfiguration(setOf(
                R.id.nav_characters, R.id.nav_comics, R.id.nav_events,
R.id.favoriteCharacterListFragment), drawerLayout
        )
        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {

        return true
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.nav_host_fragment)

        return navController.navigateUp(appBarConfiguration) || super.onSupportNavigateUp()
    }

}
```

```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/nav_characters"
        android:icon="@drawable/baseline_person_18"
        android:title="@string/menu_characters" />
    <item
        android:id="@+id/nav_comics"
        android:icon="@drawable/baseline_auto_stories_black_18"
        android:title="@string/menu_comics" />
    <item
        android:id="@+id/nav_events"
        android:icon="@drawable/baseline_event_black_18"
        android:title="@string/menu_events" />
</group>
```

RecyclerView - 1

I personaggi, i fumetti e gli eventi sono presentati tramite RecyclerView, ottimizzate per la visualizzazione di raccolte di grandi dimensioni.

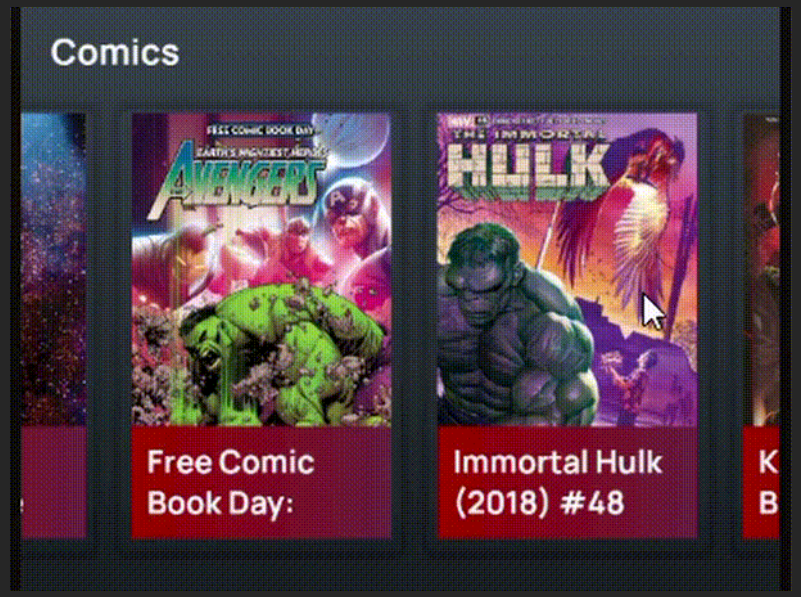
Attraverso i DataSource viene gestita la paginazione per gli “Endless” RecyclerView, poiché le API Marvel consentono di scaricare solo un numero limitato di oggetti per volta impedendone la visualizzazione completa.

```
override fun loadInitial(
    params: LoadInitialParams<Int>,
    callback: LoadInitialCallback<Int, Character>
) {
    val numberOfItems=params.requestedLoadSize
    createObservable(0,1,numberOfItems, callback, null)

}

override fun loadBefore(params: LoadParams<Int>, callback: LoadCallback<Int, Character>) {
    val page=params.key
    val numberOfItems=params.requestedLoadSize
    createObservable(page,page-1,numberOfItems, null, callback)
}

override fun loadAfter(params: LoadParams<Int>, callback: LoadCallback<Int, Character>) {

    val page=params.key
    val numberOfItems=params.requestedLoadSize
    createObservable(page,page+1,numberOfItems, null, callback)
}
```

RecyclerView - 2

Inoltre, l'utilizzo di RecyclerView offre la possibilità di implementare layout sia orizzontali che verticali.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_comics"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="16dp"
    android:orientation="horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txt_Comics" />
```

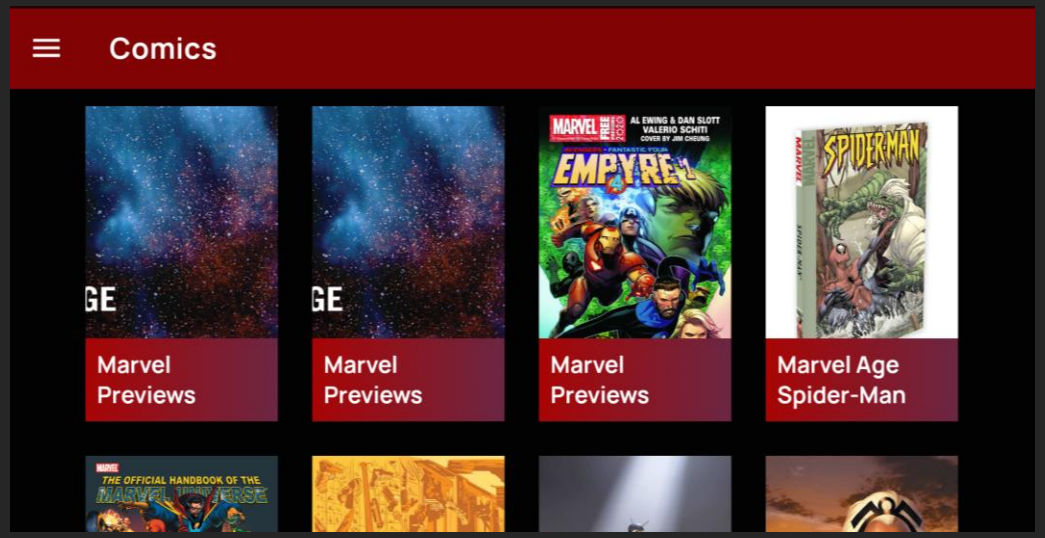


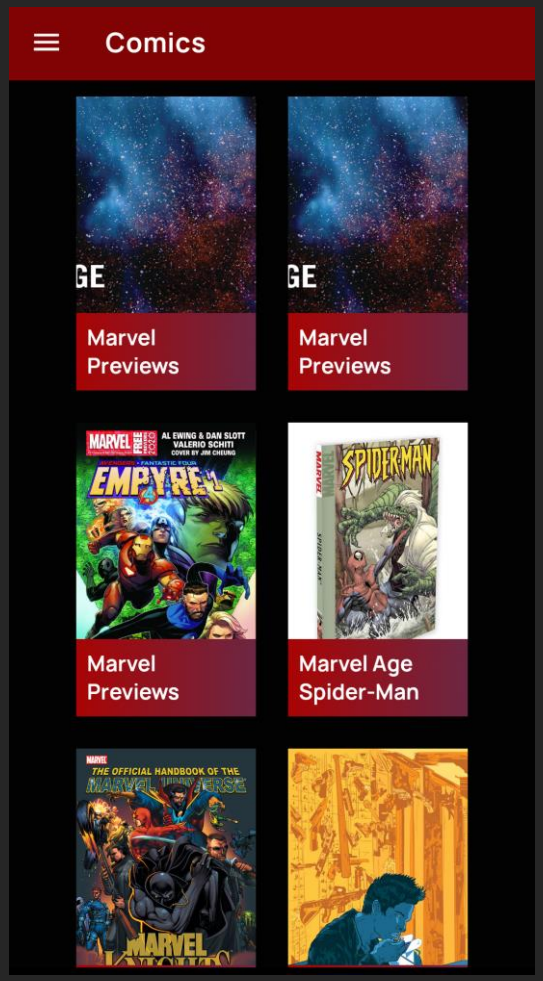
Configuration

Questa classe descrive tutte le informazioni sulla configurazione del dispositivo che possono influire sulle risorse recuperate dall'applicazione.

Ciò include sia le opzioni di configurazione specificate dall'utente e sia le configurazioni del dispositivo (come le modalità di input, le dimensioni dello schermo e l'orientamento dello schermo).

```
val llm = if(resources.configuration.orientation==Configuration.ORIENTATION_PORTRAIT) {
    GridLayoutManager(activity, 2)
}else{
    GridLayoutManager(activity, 4)
}
```





Libreria Retrofit

I personaggi, i fumetti e gli eventi sono recuperati attraverso le API messe a disposizione dalla Marvel gestite attraverso la libreria Retrofit.

```
fun getService(): MarvelAPI {
    val logging = HttpLoggingInterceptor()
    logging.level = HttpLoggingInterceptor.Level.BODY

    val httpClient = OkHttpClient.Builder()
    httpClient.addInterceptor(logging)
    httpClient.addInterceptor { chain ->
        val original = chain.request()
        val originalHttpUrl = original.url

        val ts = (Calendar.getInstance(TimeZone.getTimeZone("UTC")).timeInMillis / 1000L).toString()
        val url = originalHttpUrl.newBuilder()
                .addQueryParameter("apikey", API_KEY)
                .addQueryParameter("ts", ts)
                .addQueryParameter("hash", "$ts$PRIVATE_KEY$API_KEY".md5())
                .build()

        chain.proceed(original.newBuilder().url(url).build())
    }

    val gson = GsonBuilder().setLenient().create()
    val retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
            .addConverterFactory(GsonConverterFactory.create(gson))
            .client(httpClient.build())
            .build()

    return retrofit.create(MarvelAPI::class.java)
}
```

Estensioni

Sono state create:

- l'estensione ".load" per le ImageView:

  Attraverso la libreria Glide, carica un'immagine nell'ImageView tramite l'URL.

```
fun ImageView.load(url:String){
    Glide.with(context)
        .load(url)
        .into(this)
}
```

- l'estensione ".md5" per le stringhe:

  Data una stringa, crea il rispettivo hash md5 utile per la chiamata alle API Marvel.

```
fun String.md5():String{
    val digest=java.security.MessageDigest.getInstance("MD5")
    digest.update(toByteArray())
    val messageDigest=digest.digest()
    val hexString=StringBuilder()
    for (aMessageDigest in messageDigest){
        var h=Integer.toHexString(0xFF and aMessageDigest.toInt())
        while (h.length<2)
            h="0$h"
        hexString.append(h)
    }
    return hexString.toString()
}
```

Configurazione sicurezza di rete

A partire da Android 9, il traffico in chiaro (HTTP non crittografato) viene bloccato per impostazione predefinita.

Per ovviare a ciò, nel file XML *network_security_config* sono state specificate le impostazioni di sicurezza della rete per l'applicazione, risolvendo così il problema per il recupero delle immagini dei personaggi dal sito Marvel.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">i.annihil.us</domain>
    </domain-config>
</network-security-config>
```

Per il corretto funzionamento dell'app, una volta specificate le impostazioni di rete, il file XML è stato poi richiamato nel Manifest:

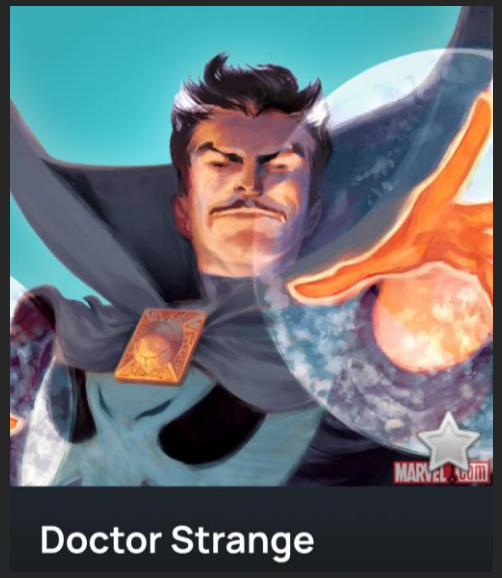
```
android:networkSecurityConfig="@xml/network_security_config"
```

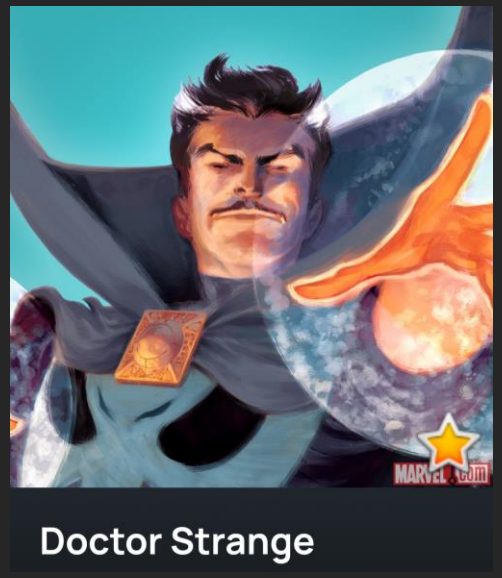
Favorite Characters

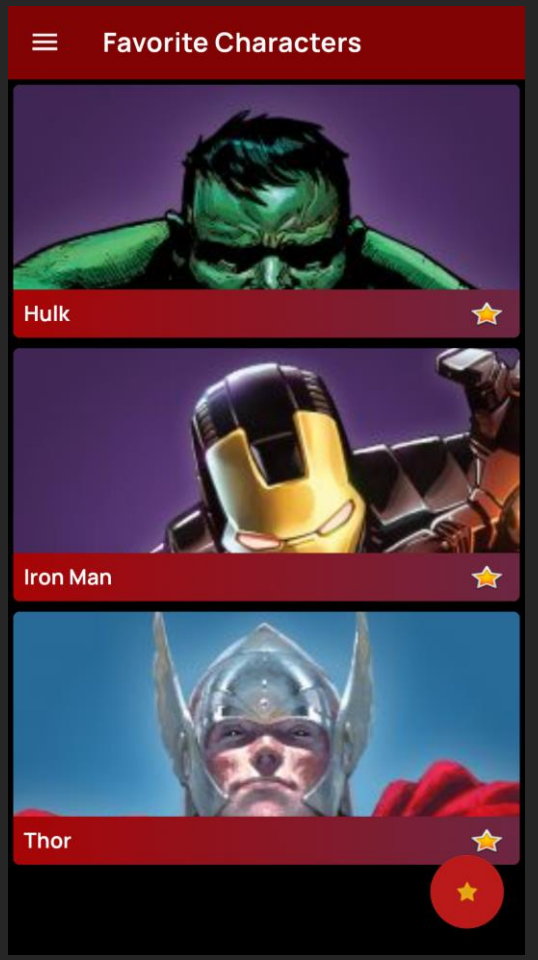
Come precedentemente accennato, cliccando sul Floating Action Button si accede al Fragment dedicato ai Personaggi Preferiti.

L’inserimento o la rimozione dalla Lista Preferiti avviene spuntando la stellina accanto al nome del personaggio.

In caso di aggiunta la stellina passa da grigia a gialla, viceversa in caso di rimozione.

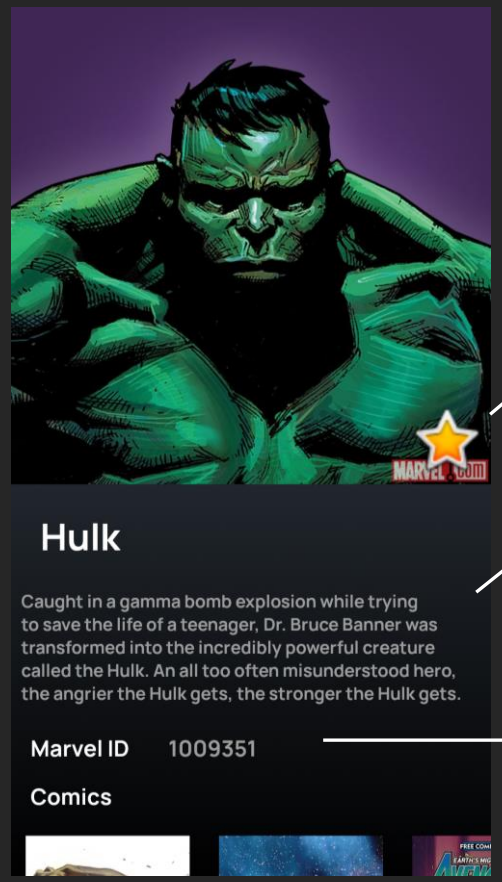






Visualizzazione Personaggio

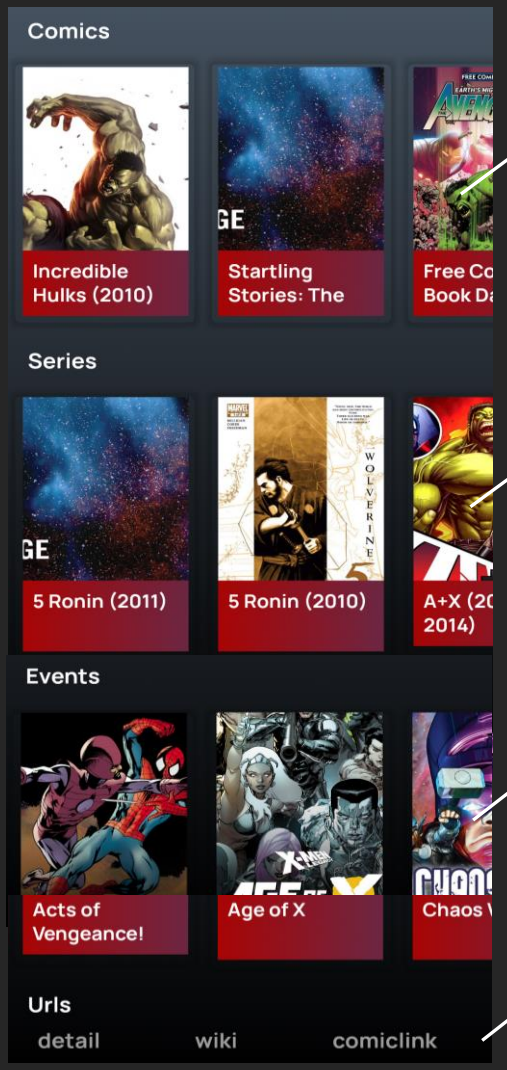
Cliccando su un Personaggio si accede alla CharacterDatailsActivity, ovvero alla pagina dedicata al personaggio selezionato.



Stellina dei Preferiti

Descrizione Personaggio

Codice Identificativo Personaggio



Sezione Fumetti

Sezione Serie

Sezione Eventi

Sezione Link Web

Sezione Link Web - 1

La gestione degli URL relativi al Personaggio scelto avviene nel modo seguente:

```
val lenght=character.urls.size
if(lenght>0)
    binding.txtDetail.text = character.urls[0].type
if(lenght>1)
    binding.txtWiki.text = character.urls[1].type
if(lenght>2)
    binding.txtComiclink.text = character.urls[2].type

binding.txtDetail.setOnClickListener{
    val intent = Intent(Intent.ACTION_VIEW)
    intent.data = Uri.parse(character.urls[0].url)
    startActivity(intent)
}

binding.txtWiki.setOnClickListener{
    val intent = Intent(Intent.ACTION_VIEW)
    intent.data = Uri.parse(character.urls[1].url)
    startActivity(intent)
}

binding.txtComiclink.setOnClickListener{
    val intent = Intent(Intent.ACTION_VIEW)
    intent.data = Uri.parse(character.urls[2].url)
    startActivity(intent)
}
```

Tramite il costrutto if viene effettuato il controllo sul numero di link associati al personaggio e creato il collegato tra TextView e URL.

In questo modo, cliccando su una delle TextView a disposiozione tra “Detail”, “Wiki” e “ComicLink”, si viene automaticamente rimandati alla relativa pagina web.

Per eseguire queste operazioni di rete è stata inclusa nel Manifest la seguente autorizzazione:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Sezione Link Web - 2