

MARVEL PLANET

Splashscreen



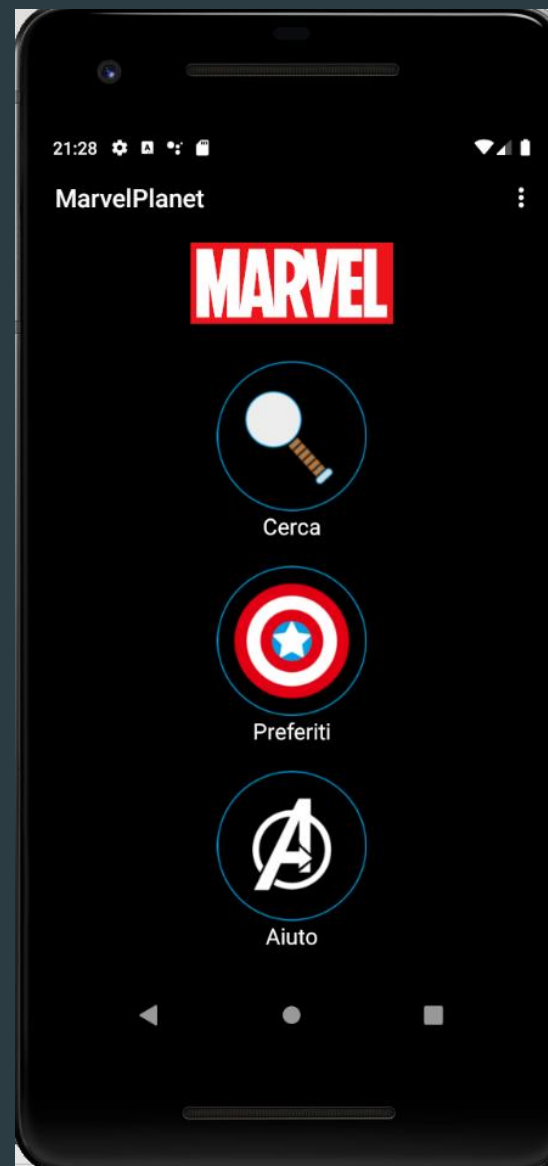
- ▶ Al momento del lancio dell'applicazione, apparirà per un tempo di 2,3 secondi uno splashscreen.

```
int SPLASH_DISPLAY_LENGTH = 2300;
new Handler().postDelayed(new Runnable(){
    @Override
    public void run() {
        Intent mainIntent = new Intent( packageContext: splash.this, MainActivity.class);
        splash.this.startActivity(mainIntent);
        splash.this.finish();
    }
}, SPLASH_DISPLAY_LENGTH);
```

HOME PAGE

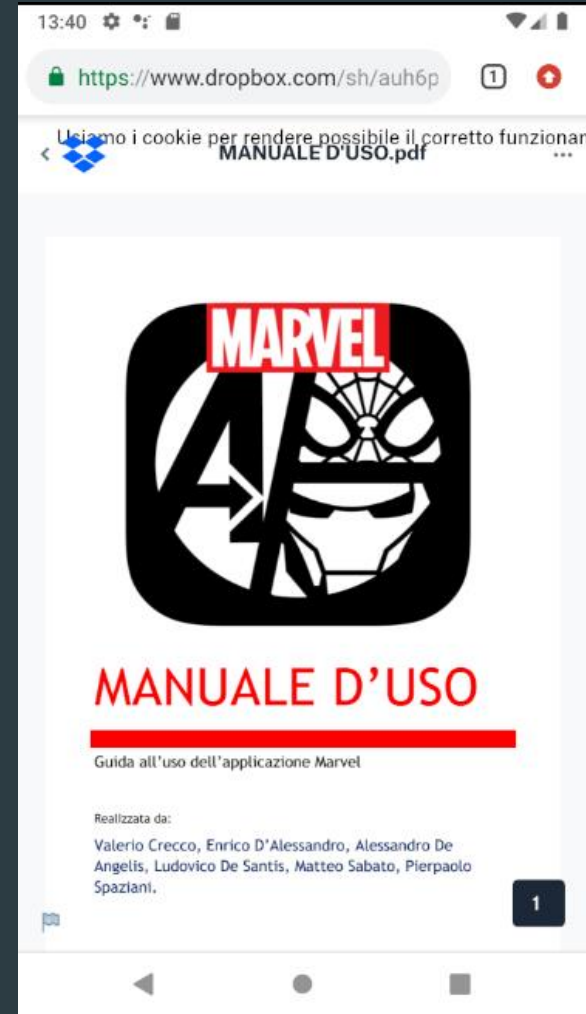
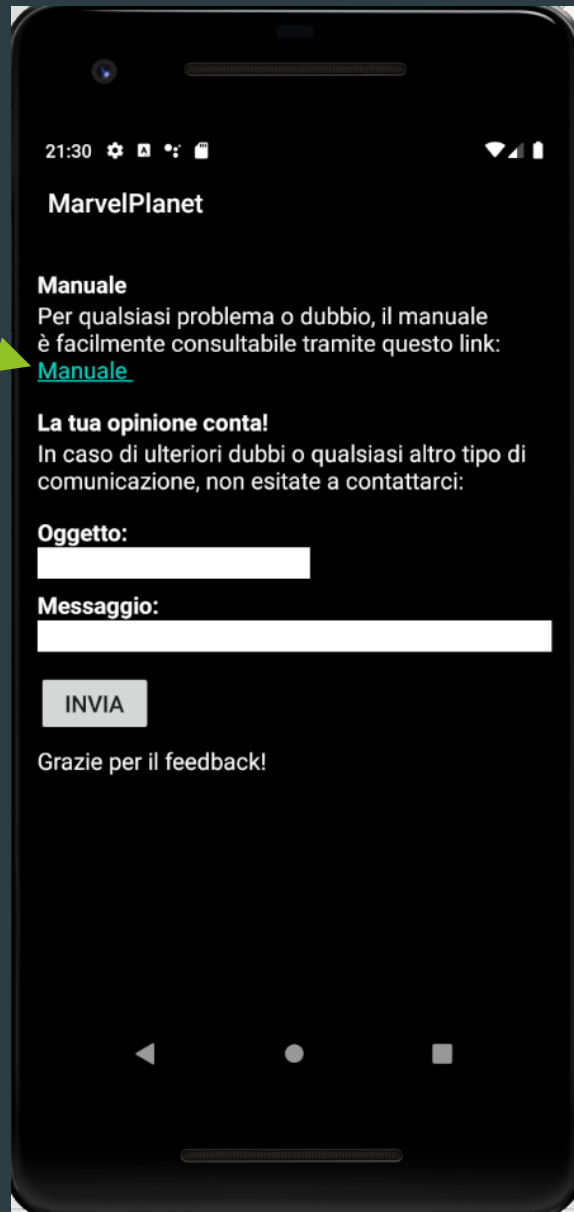
Dalla pagina iniziale si può facilmente accedere alle seguenti sezioni:

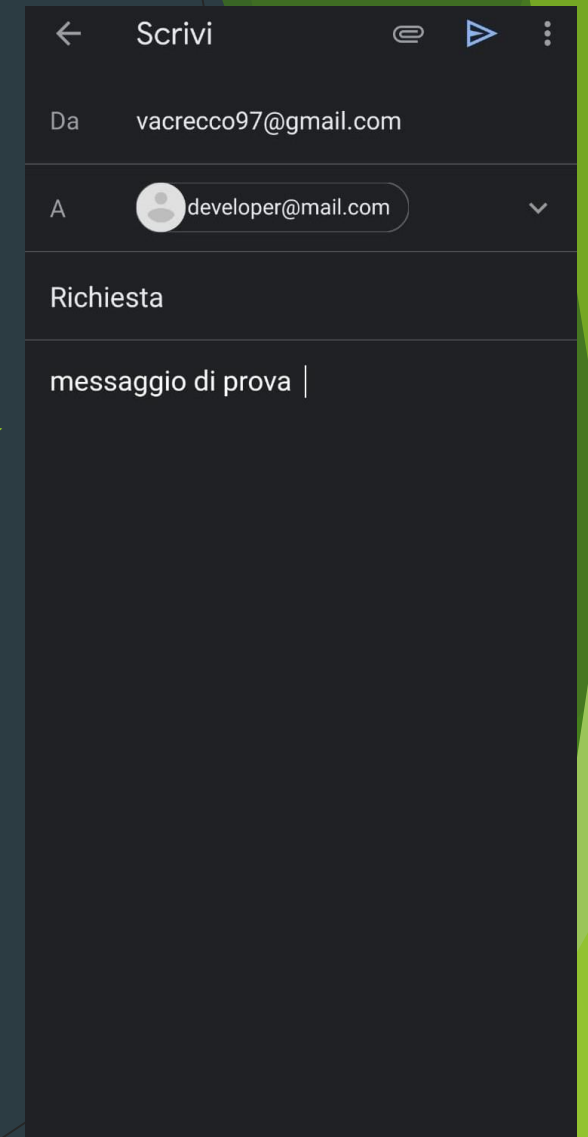
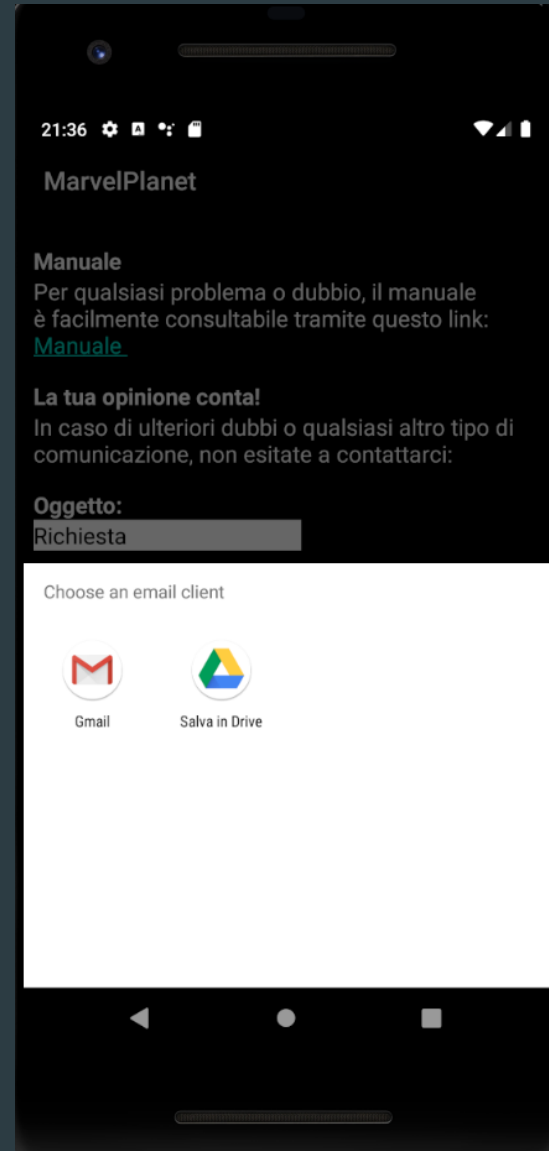
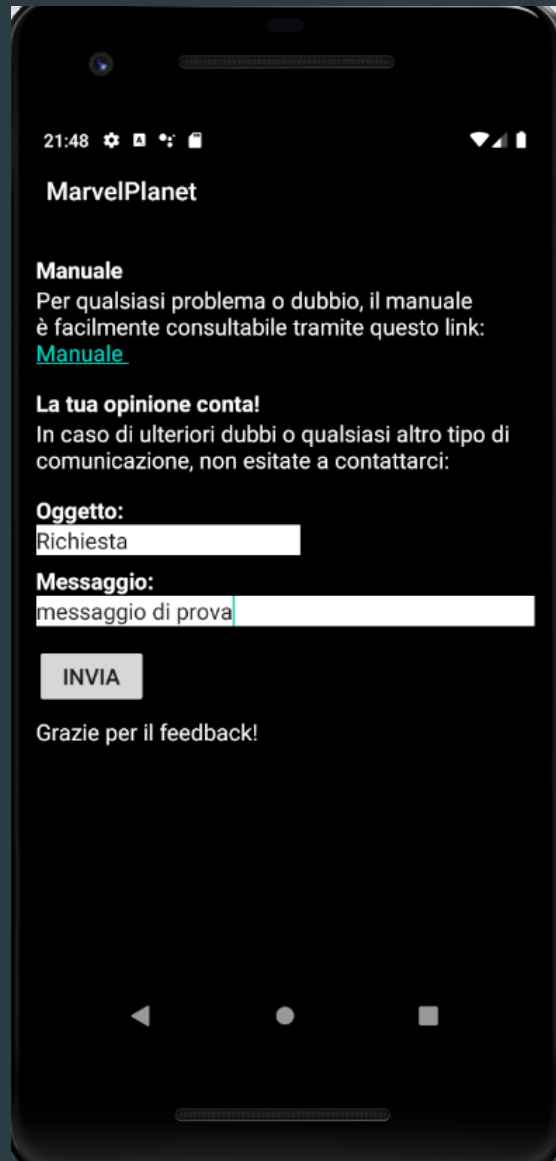
- ▶ **Cerca:** permette di cercare eroi o fumetti;
- ▶ **Preferiti:** permette di visualizzare la lista dei preferiti e le relative informazioni;
- ▶ **Aiuto:** permette di essere reindirizzati al manuale dell'applicazione o di contattare direttamente il team degli sviluppatori via mail.



Aiuto

- ▶ Questa sezione permette di essere reindirizzati direttamente al manuale online dell'applicazione.
- ▶ Permette, inoltre, di contattare il team di sviluppatori per eventuali ulteriori chiarimenti riguardo alcune funzionalità dell'app o per lasciare un feedback relativo all'app stessa.
- ▶ Una volta riempiti i campi «oggetto» e «messaggio», premendo il tasto «invia» si potrà scegliere un client e-mail per poter inviare la mail utilizzando l'account predisposto, trovando già inserito nel campo «destinatario» la mail del team di sviluppatori oltre alle informazioni già inserite nell'app.

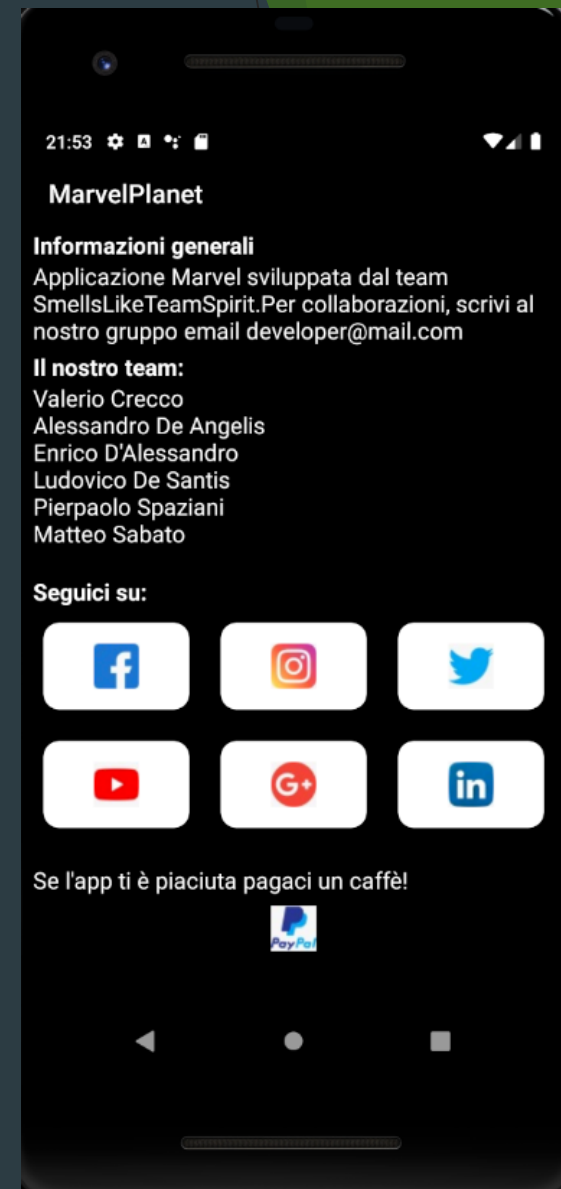
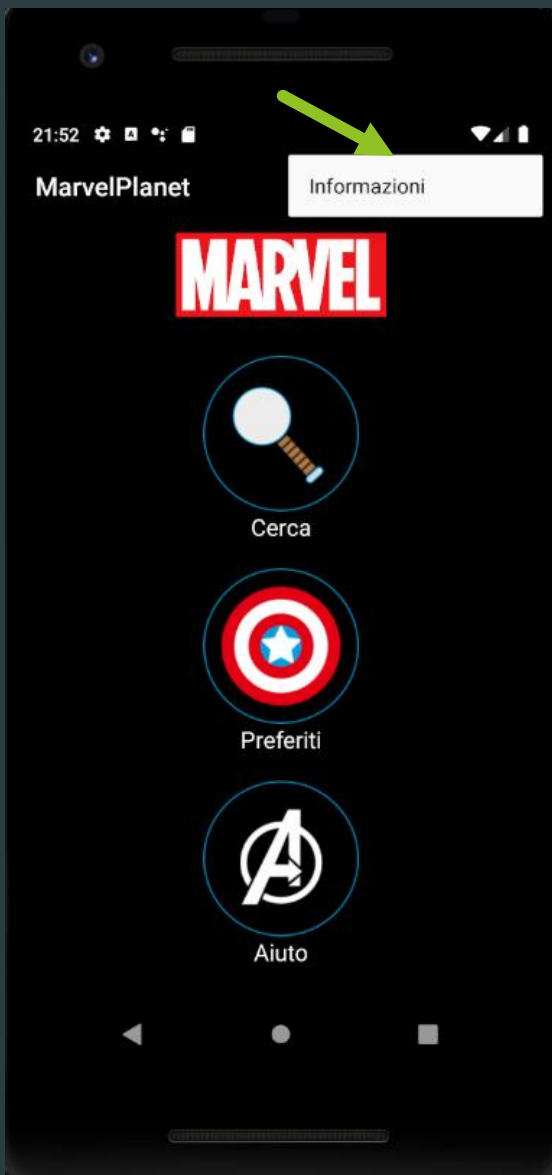
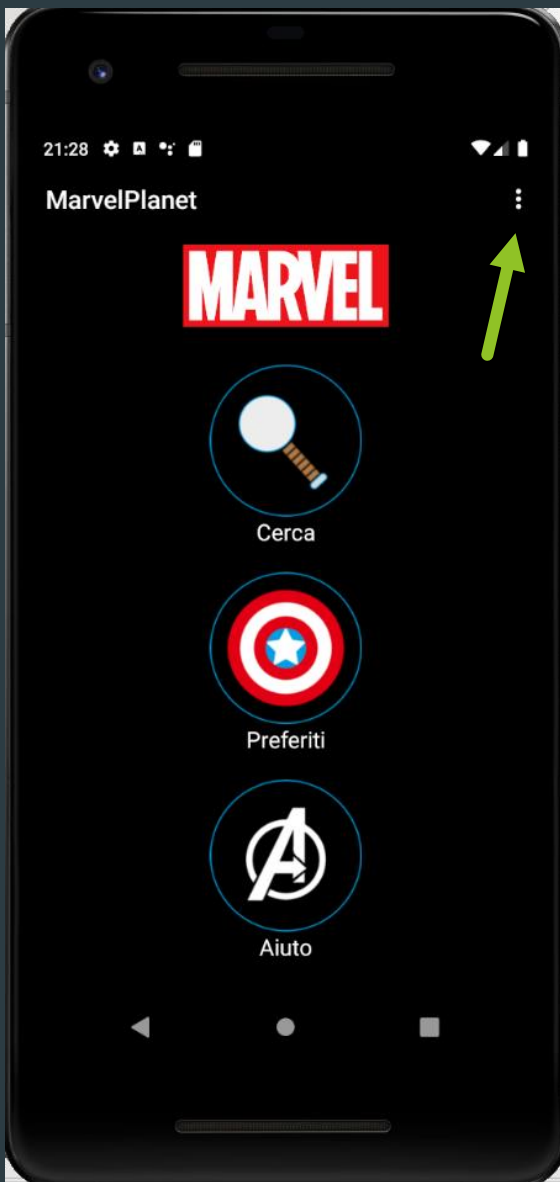




Informazioni

- ▶ Questa sezione, accessibile come mostrato nella foto successiva, mostra la composizione del team di sviluppatori, l'indirizzo mail per effettuare richieste di collaborazioni e i link ai profili social relativi, oltre ad un link Paypal per eventuali donazioni.
- ▶ Estratto di codice per accedere al profilo Instagram;

```
if(v.getId() == R.id.ib_instagram) {  
    Intent intent = new Intent();  
    intent.setAction(Intent.ACTION_VIEW);  
    intent.addCategory(Intent.CATEGORY_BROWSABLE);  
    intent.setData(Uri.parse("https://www.instagram.com/?hl=it"));  
    startActivity(intent);  
}
```



The background features a dark blue-grey field on the left, transitioning into a series of overlapping, semi-transparent green and yellow-green geometric shapes on the right. These shapes are primarily triangles and polygons, creating a layered, abstract effect.

VOLLEY & JSON

Volley

- ▶ L'applicazione utilizza come fonte di dati il database messo a disposizione dalla Marvel.
- ▶ Vengono inoltre fornite delle API per l'interrogazione di tale database.
- ▶ Il risultato che si ottiene come risposta può essere scelto in diversi formati.
 - ▶ Il formato scelto in questo caso è quello JSON.

Volley

- ▶ Le richieste di rete vengono effettuate grazie alla libreria Volley.
- ▶ La scelta di tale libreria è dovuta alla sua semplicità e velocità rispetto ad altre soluzioni.
- ▶ Inoltre la mole di dati in download dopo una certa request, qualsiasi essa sia, non è esageratamente grande.
 - ▶ Anche se composta da un numero elevato di «oggetti», le dimensioni effettive del file JSON sono molto limitate (quasi irrisorie a volte).

Volley

- ▶ La classe fondamentale di Volley è la RequestQueue, nella quale vengono messe in coda (FIFO) tutte le richieste.
- ▶ Si fa notare che in questa applicazioni non si presenteranno mai scenari in cui sia possibile effettuare richieste multiple, ciò implica un carico di lavoro molto basso
- ▶ Viene quindi allocata una requestQueue, pronta per essere utilizzata come coda delle richieste da effettuare.

Volley

- ▶ Le richieste che abbiamo citato prima sono degli oggetti di tipo Request.
- ▶ La classe Request è utilizzata per creare istanze di accesso alla rete che supportano richieste di tipo GET e POST.
- ▶ Volley fornisce diversi tipi di Request, quelle utilizzate in tale applicazione sono la StringRequest e l'ImageRequest.

StringRequest

- ▶ Nella StringRequest viene specificata l'URL dal quale ricevere poi la risposta.
- ▶ La risposta ad una StringRequest consiste in un file JSON
- ▶ Tale file JSON viene consegnato al metodo *onResponse(String response)* come una stringa che viene poi trasformata in un oggetto di tipo JSONObject.

StringRequest

- ▶ Il metodo *SearchHeroesByName(String s)* in figura si occupa di creare una *StringRequest* per ottenere un personaggio di cui si fa richiesta esplicita specificando il nome.
- ▶ Il nome viene passato al metodo come parametro e inserito nell'URL insieme al Time Stamp, l'Api Key ottenuta dalla piattaforma Marvel e un codice HASH
- ▶ La richiesta creata viene inserita nella *RequestQueue* creata in precedenza.

```
public void searchHeroesByName(String s) {  
    String url = "https://gateway.marvel.com/v1/public/characters?ts=%s&name=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, s, APIKEY, HASH);  
    Log.w( tag: "TAG_M", s);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, listener: this, error -> {});  
    requestQueue.add(stringRequest);  
}
```

ImageRequest

- ▶ La richiesta di un'immagine viene fatta dal metodo
getImage(int position, String strHeroThumb, final ImageView imageView)
- ▶ A questo metodo viene passata la posizione nella recyclerView, l'URL dell'immagine e l'oggetto imageView alla quale inserire la foto richiesta
- ▶ Nell'ImageRequest viene specificato l'URL dell'immagine da scaricare.
- ▶ Per le ImageRequest la risposta consiste in un oggetto di tipo Bitmap.

```
private void getImage(int position, String strHeroThumb, final ImageView imageView) {  
    requestQueue.cancelAll(position);  
    ImageRequest imageRequest = new ImageRequest(strHeroThumb,  
        imageView::setImageBitmap, maxWidth: 0, maxHeight: 0, scaleType: null, Bitmap.Config.ARGB_8888,  
        error -> imageView.setImageResource(R.drawable.ic_launcher_background));  
    imageRequest.setTag(position);  
    requestQueue.add(imageRequest);  
}
```


Parse JSON

- ▶ Abbiamo visto che il file JSON consegnato al metodo `onResponse(String response)` di Volley è una stringa.
- ▶ Nella classe «HeroAPI» questa stringa per prima cosa è stata trasformata in un oggetto di tipo `JSONObject`:
«`JSONObject jsonObjectResponse = new JSONObject(response)`».
- ▶ Al lato è mostrato un risultato ottenuto interrogando il database messo a disposizione dalla Marvel grazie all'utilizzo dell'API per la ricerca di un eroe.

```
{
  "code": 200,
  "status": "Ok",
  "copyright": "© 2020 MARVEL",
  "attributionText": "Data provided by Marvel. © 20",
  "attributionHTML": "<a href=\"http://marvel.com\"",
  "etag": "1ea744adacc05dede0624a161af1664397a92faf",
  "data": {
    "offset": 0,
    "limit": 20,
    "total": 1,
    "count": 1,
    "results": [
      {
        "id": 1009351,
        "name": "Hulk",
        "description": "Caught in a gamma bomb",
        "modified": "2020-07-21T10:35:15-0400",
        "thumbnail": { ... }, // 2 items
        "resourceURI": "http://gateway.marvel.",
        "comics": { ... }, // 4 items
        "series": { ... }, // 4 items
        "stories": { ... }, // 4 items
        "events": { ... }, // 4 items
        "urls": [ ... ] // 3 items
      }
    ]
  }
}
```

Parse JSON

- I dati di nostro interesse si trovano sotto il campo identificato dalla chiave «results». Il valore associato a questa chiave è un array di elementi complessi ciascuno rappresentante un determinato eroe.
- A sua volta «results» si trova all'interno della campo con chiave «data».

```
JSONObject jsonObjectResponse = new JSONObject(response);
JSONObject jsonObjectData = jsonObjectResponse.getJSONObject("data");
JSONArray jsonArrayResults = jsonObjectData.getJSONArray("results");
```

- In questo esempio di risposta possiamo vedere che sono stati restituiti 8 eroi differenti.

[illegible]

Parse JSON

- E' stata definita un'entità «Hero» nella quale sono stati inseriti diversi attributi tra i quali l'id dell'eroe, il suo nome, una sua descrizione e altre informazioni.

```
▼ "results": [  
  ▼ {  
    "id": 1009664,  
    "name": "Thor",  
    "description": "As the Norse God of  
oafish imbecile, he's quite smart and  
    "modified": "2020-03-11T10:18:50-0400",  
    ▶ "thumbnail": { ... }, // 2 items  
    "resourceURI": "http://gateway.marvel.com/v1/public/characters/1009664",  
    ▶ "comics": { ... }, // 4 items  
    ▶ "series": { ... }, // 4 items  
    ▶ "stories": { ... }, // 4 items  
    ▶ "events": { ... }, // 4 items  
    ▶ "urls": [ ... ] // 3 items  
  },  
]
```

```
public class Hero implements Parcelable {  
  
    private int id;  
    private String name;  
    private String description;  
    private String resURI;  
    private MarvelImage imgHero;  
    private String collectionURIComics;  
    private String collectionURISeries;  
    private String collectionURISTories;  
    private String collectionURIEvents;  
}
```

Parse JSON

- ▶ Per ogni eroe ottenuto dalla richiesta è stato istanziato un nuovo oggetto di tipo «Hero» e sono stati popolati i suoi attributi come mostrato in figura:
- ▶ Per quanto riguarda la Thumbnail dell'eroe è stata definita l'entità «MarvelImage».

```
public class MarvelImage implements Parcelable {  
  
    private String path;  
    private String ext;  
  
    public MarvelImage(String path, String ext) {  
        this.path = path;  
        this.ext = ext;  
    }  
}
```

```
JSONObject object = jsonArrayResults.getJSONObject(i);  
  
Hero hero = new Hero();  
hero.setId(object.getInt( name: "id"));  
hero.setName(object.getString( name: "name"));  
hero.setDescription(object.getString( name: "description"));  
hero.setResURI(object.getString( name: "resourceURI"));  
  
JSONObject jsonObjectThumbnail = object.getJSONObject("thumbnail");  
hero.setImgHero(new MarvelImage(jsonObjectThumbnail.getString( name: "path"),  
                                jsonObjectThumbnail.getString( name: "extension")));  
  
JSONObject jsonObjectComics = object.getJSONObject("comics");  
hero.setCollectionURIComics(jsonObjectComics.getString( name: "collectionURI"));  
  
JSONObject jsonObjectSeries = object.getJSONObject("series");  
hero.setCollectionURISeries(jsonObjectSeries.getString( name: "collectionURI"));  
  
JSONObject jsonObjectStories = object.getJSONObject("stories");  
hero.setCollectionURISTories(jsonObjectStories.getString( name: "collectionURI"));  
  
JSONObject jsonObjectEvents = object.getJSONObject("events");  
hero.setCollectionURIEvents(jsonObjectEvents.getString( name: "collectionURI"));  
  
heroes.add(hero);
```


MarvelImage

- ▶ Il Marvel Developer Portal mette a disposizione diversi formati di immagini che è possibile richiedere attraverso le API.
- ▶ All'interno dell'entità «MarvelImage» sono stati definiti due metodi per la richiesta di immagini in base alla dimensione desiderata.

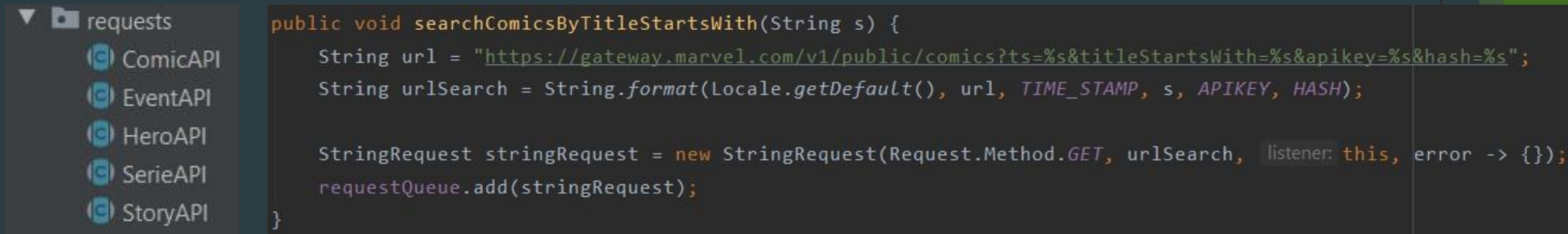
```
public String getFullPathMedium() {  
    String fullPath = path + "/standard_xlarge" + "." + ext;  
    return fullPath;  
}  
  
public String getFullPathLarge(){  
    String fullPath = path + "/standard_fantastic" + "." + ext;  
    return fullPath;  
}
```

```
ImageRequest request = new ImageRequest(hero.getImgHero().getFullPathLarge(),
```

standard_small	65x45px
standard_medium	100x100px
standard_large	140x140px
standard_xlarge	200x200px
standard_fantastic	250x250px
standard_amazing	180x180px

Parse JSON e altre API

- Nel package «requests» troviamo anche le classi per l'utilizzo di altre API, come quelle inerenti ai fumetti, eventi, serie e storie.

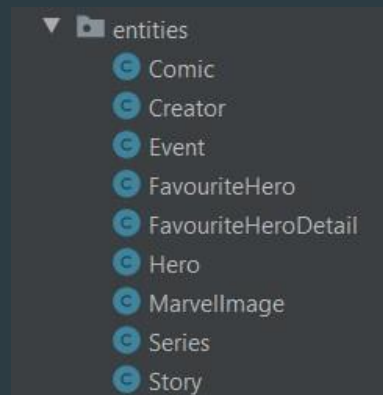


```
▼ requests
  (C) ComicAPI
  (C) EventAPI
  (C) HeroAPI
  (C) SerieAPI
  (C) StoryAPI

public void searchComicsByTitleStartsWith(String s) {
    String url = "https://gateway.marvel.com/v1/public/comics?ts=%s&titleStartsWith=%s&apikey=%s&hash=%s";
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, s, APIKEY, HASH);

    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, listener: this, error -> {});
    requestQueue.add(stringRequest);
}
```

- Il parse del file JSON ottenuto come risposta è stato realizzato in modo analogo a quanto visto precedentemente definendo le entità comic, event, serie e story.



```
▼ entities
  (C) Comic
  (C) Creator
  (C) Event
  (C) FavouriteHero
  (C) FavouriteHeroDetail
  (C) Hero
  (C) MarvellImage
  (C) Series
  (C) Story
```

LIST VIEW

Search Hero

- Dopo aver selezionato **Heroes** nella pagina di selezione della ricerca verrà mostrata una schermata dove è possibile cercare gli eroi, potendo anche scegliere se filtrare o meno la ricerca con il nome completo dell'eroe ...



Search Comic

- ... oppure, selezionando Comics, si può effettuare la ricerca dei fumetti



Come?

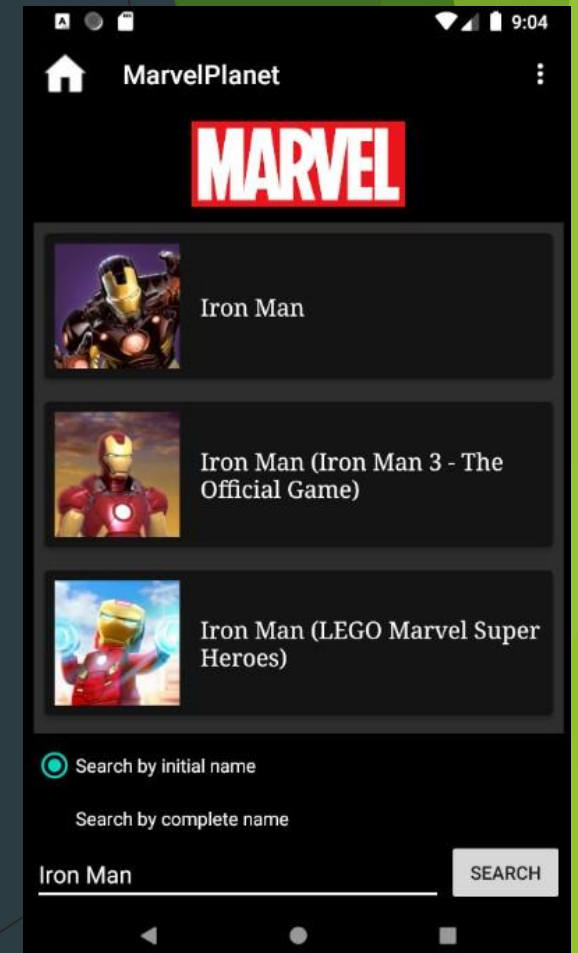
- La pagina di ricerca degli eroi viene gestita dal **FragmentHeroList**
- Nel momento in cui viene premuto il tasto *Search*, viene fatta una ricerca grazie alle API fornite dal *marvel developer portal* e viene popolata la lista degli eroi dell'**HeroListViewModel** che viene passata all'**HeroAdapter**

- Se il radio button è impostato sul nome completo viene eseguita la request:

```
public void searchHeroesByName(String s) {  
    String url = "https://gateway.marvel.com/v1/public/characters?ts=%s&name=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, s, APIKEY, HASH);  
    Log.w("TAG_M", s);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, this, error -> {});  
    requestQueue.add(stringRequest);  
}
```

- Altrimenti:

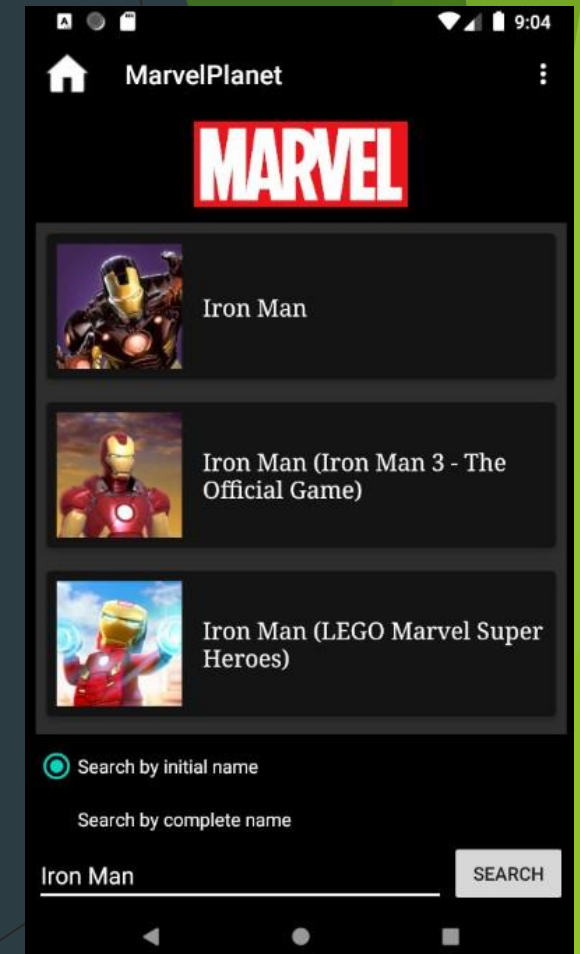
```
public void searchHeroesByNameStartsWith(String s) {  
    String url = "https://gateway.marvel.com/v1/public/characters?ts=%s&nameStartsWith=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, s, APIKEY, HASH);  
    Log.w("TAG_M", s);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, this, error -> {});  
    requestQueue.add(stringRequest);  
}
```



Come?

- La *RecyclerView* viene quindi riempita dall'*HeroAdapter* con le *CardView*

```
static class Holder extends RecyclerView.ViewHolder {  
    final TextView tvHero;  
    final ImageView ivHero;  
    ConstraintLayout clContainer;  
  
    Holder(@NonNull View itemView) {  
        super(itemView);  
        tvHero = itemView.findViewById(R.id.tvHero);  
        ivHero = itemView.findViewById(R.id.ivHero);  
        clContainer = itemView.findViewById(R.id.cvlItemHeroLayout);  
    }  
}  
  
public Holder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
    CardView cl;  
    cl = (CardView) LayoutInflater.from(parent.getContext()).inflate(R.layout.layout_character, parent, false);  
    cl.setOnClickListener(this);  
    return new Holder(cl);  
}  
  
public void onBindViewHolder(@NonNull Holder holder, int position) {  
    Hero hero = heroes.get(position);  
    holder.tvHero.setText(hero.getName());  
    getImage(position, hero.getImgHero().getFullPathMedium(), holder.ivHero);  
}
```



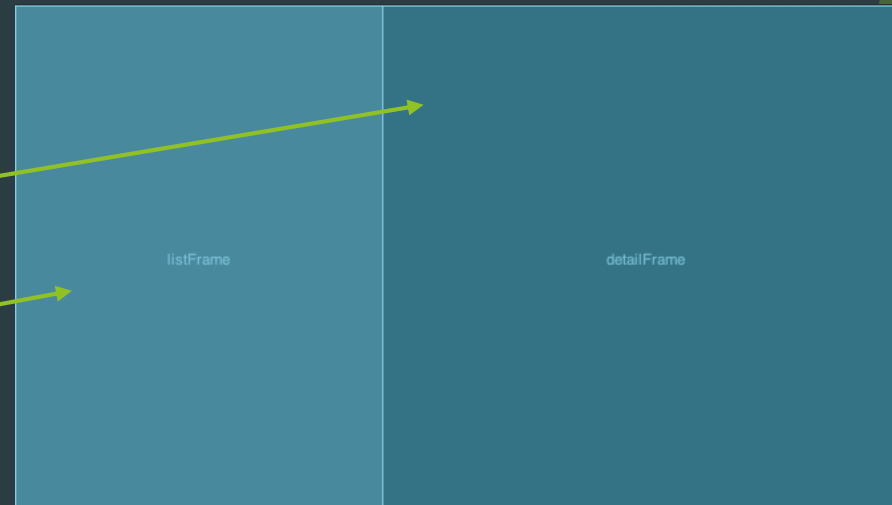
Fragment

- Nell'applicazione sono stati utilizzati i *Fragment*
- In particolare nella sezione di ricerca degli eroi, è l'activity **SearchHero** che li gestisce

```
private void portrait() {  
    fragmentHeroDetail = new FragmentHeroDetail();  
    fragmentHeroList = new FragmentHeroList(this::switchFragment);  
    removeFragments();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroList)  
        .commit();  
}
```

```
private void landscape() {  
    fragmentHeroDetail = new FragmentHeroDetail();  
    fragmentHeroList = new FragmentHeroList();  
    removeFragments();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.detailFrame, fragmentHeroDetail)  
        .commit();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroList)  
        .commit();  
}
```

```
void switchFragment() {  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroDetail)  
        .addToBackStack("MASTER")  
        .commit();  
}
```



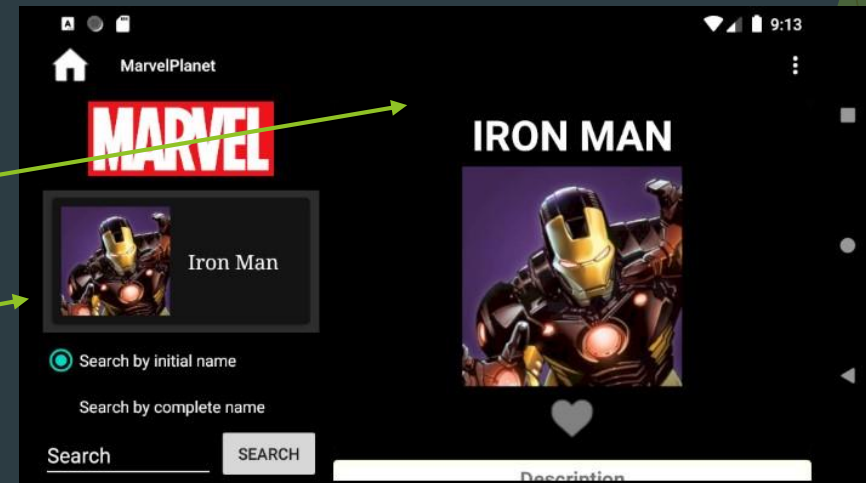
Fragment - land

- Nella versione *landscape*:
 - sulla *sinistra* viene utilizzato il **FragmentHeroList**
 - sulla *destra* viene utilizzato il **FragmentHeroDetail**

```
private void portrait() {  
    fragmentHeroDetail = new FragmentHeroDetail();  
    fragmentHeroList = new FragmentHeroList(this::switchFragment);  
    removeFragments();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroList)  
        .commit();  
}
```

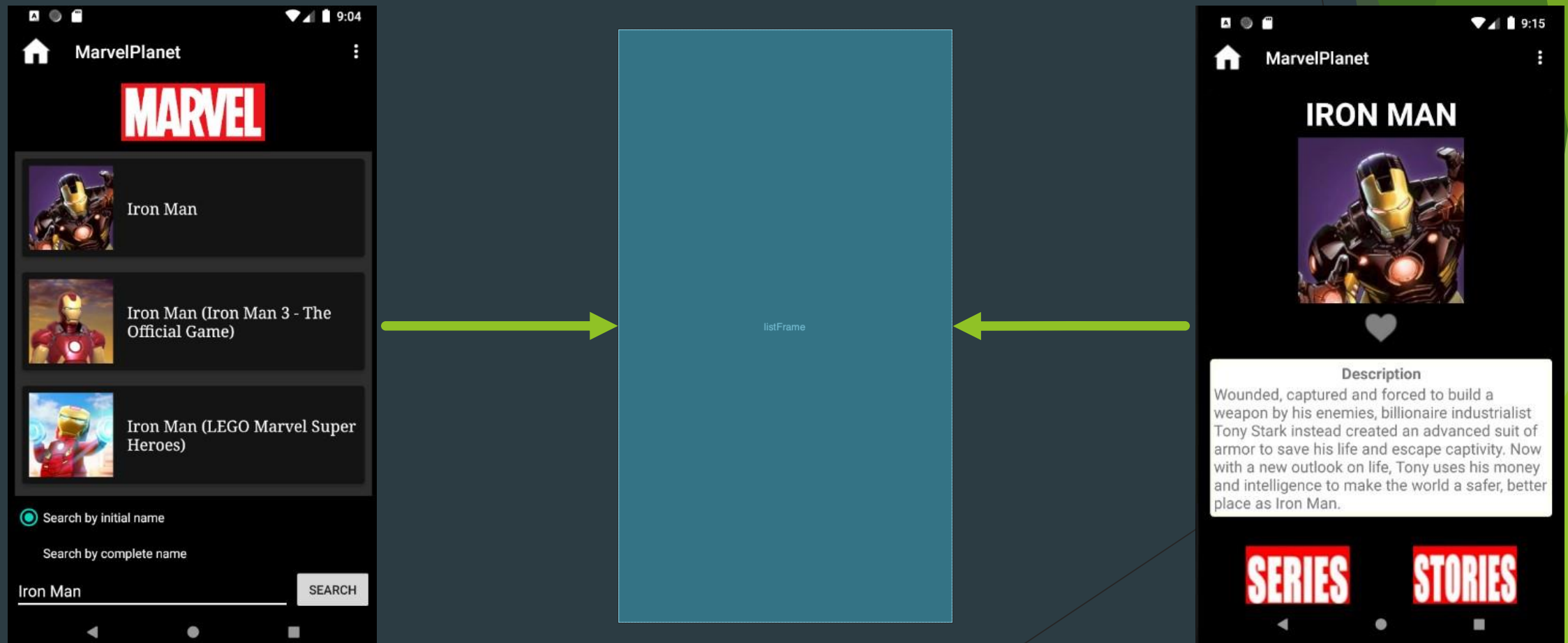
```
private void landscape() {  
    fragmentHeroDetail = new FragmentHeroDetail();  
    fragmentHeroList = new FragmentHeroList();  
    removeFragments();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.detailFrame, fragmentHeroDetail)  
        .commit();  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroList)  
        .commit();  
}
```

```
void switchFragment() {  
    getSupportFragmentManager()  
        .beginTransaction()  
        .replace(R.id.listFrame, fragmentHeroDetail)  
        .addToBackStack("MASTER")  
        .commit();  
}
```



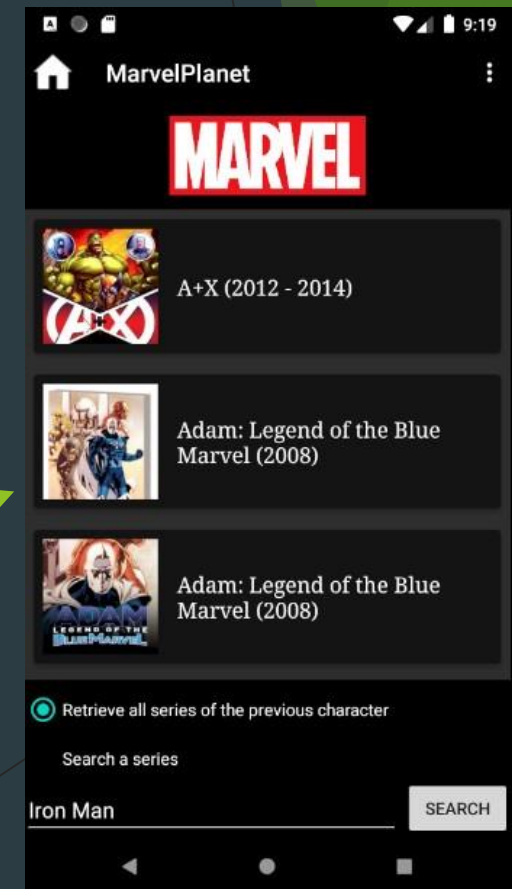
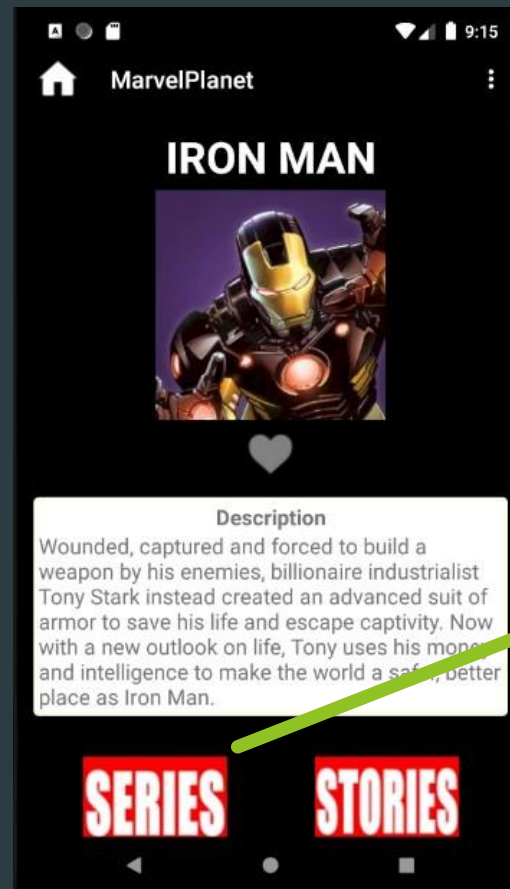
Fragment - portrait

- Nella versione *portrait* invece il **FragmentHeroList** e il **FragmentHeroDetail** vengono intercambiati sullo stesso *FrameLayout*



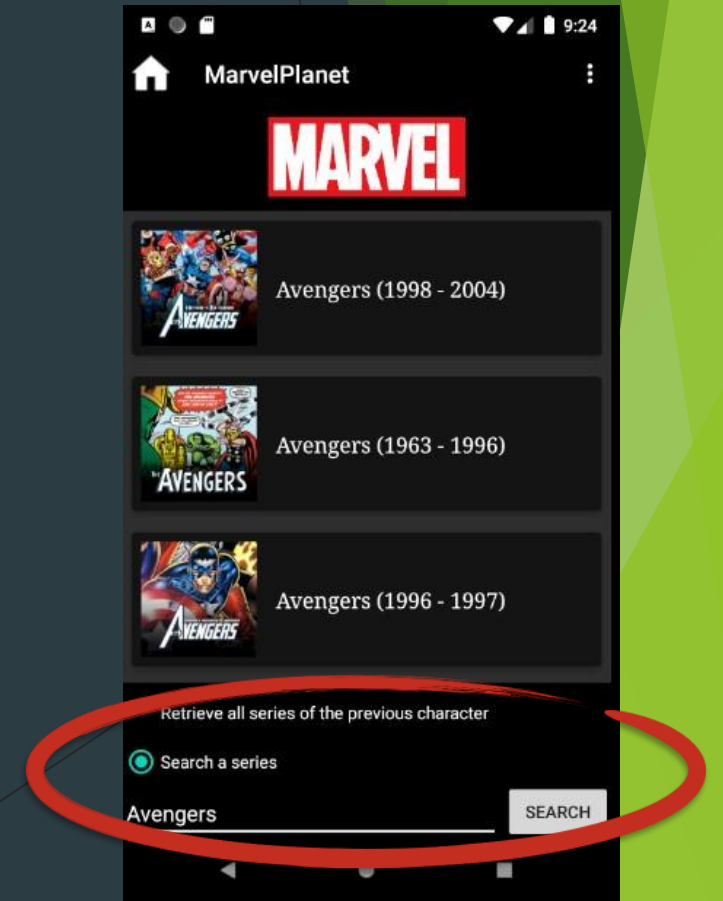
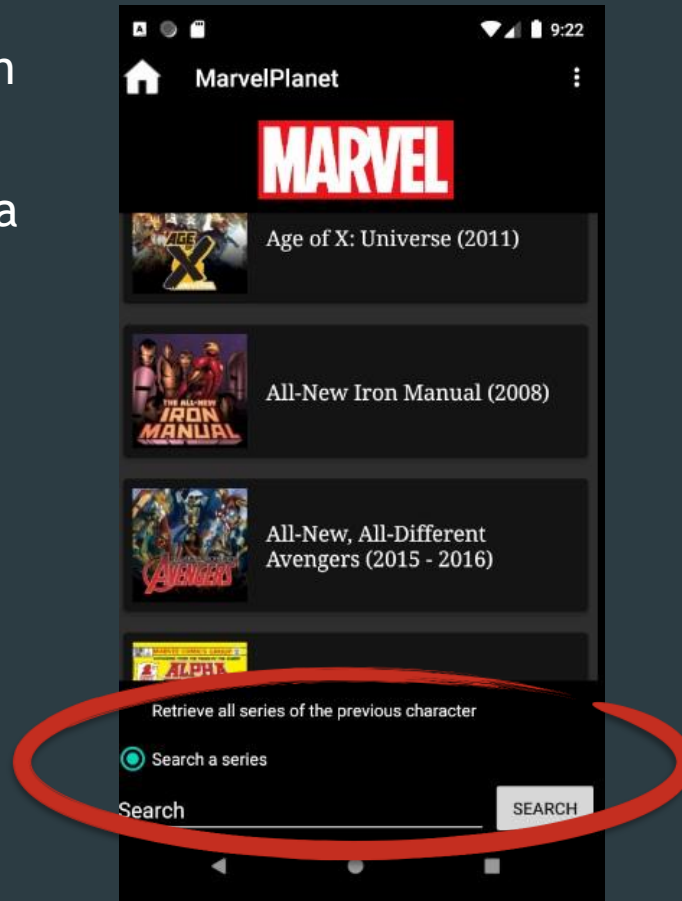
Series

- Utilizzando il bottone **Series** nella pagina di dettaglio dell'eroe, è possibile accedere alla sezione delle *serie*
- Anche in questa pagina di ricerca ci sono 2 *RadioButton* che permettono di filtrare la ricerca
- Il *primo* permette di trovare le *serie* relative all'eroe precedentemente selezionato



Series

- Utilizzando il bottone **Series** nella pagina di dettaglio dell'eroe, è possibile accedere alla sezione delle *serie*
- Il *secondo* può essere utilizzato in due modi:
 - Se viene effettuata la ricerca senza inserire alcun testo verrà mostrato l'elenco completo delle serie disponibili
 - Se invece viene digitato il nome di una serie la ricerca sarà relativa ad essa



Series

- Ciò che differenzia i 3 tipi di ricerca sono le request che vengono eseguite

- Nel caso di ricerca delle serie dell'eroe:

```
public void searchSeriesByHeroID(String s) {  
    String url = "https://gateway.marvel.com/v1/public/characters/%s/series?ts=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, s, TIME_STAMP, APIKEY, HASH);  
    Log.w("TAG_M", s);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, this, error -> {});  
    requestQueue.add(stringRequest);  
}
```

- Nel caso di ricerca di tutte le serie:

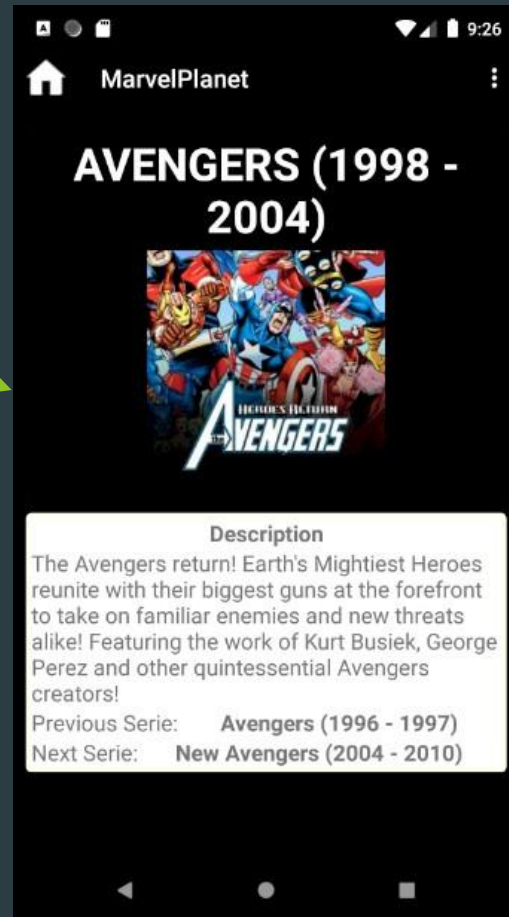
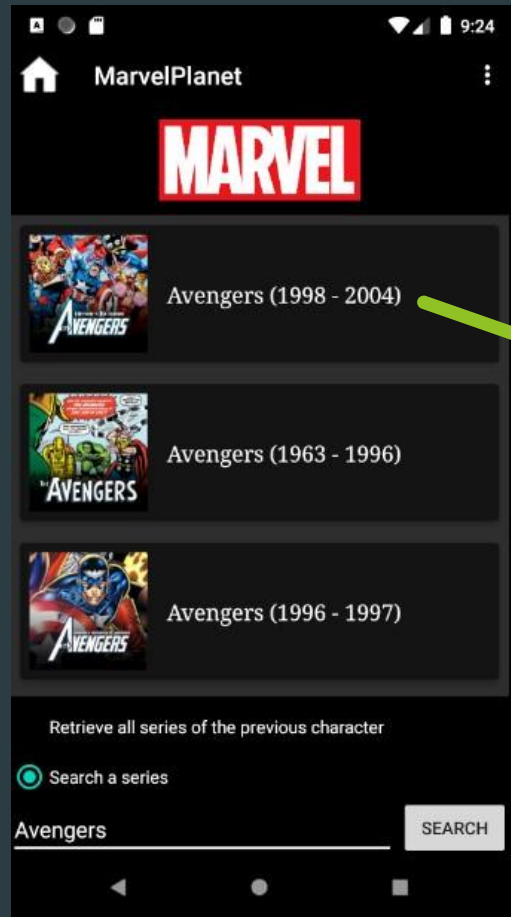
```
public void searchSeries() {  
    String url = "https://gateway.marvel.com/v1/public/series?ts=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, APIKEY, HASH);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, this, error -> {});  
    requestQueue.add(stringRequest);  
}
```

- Nel caso di ricerca di una serie specifica:

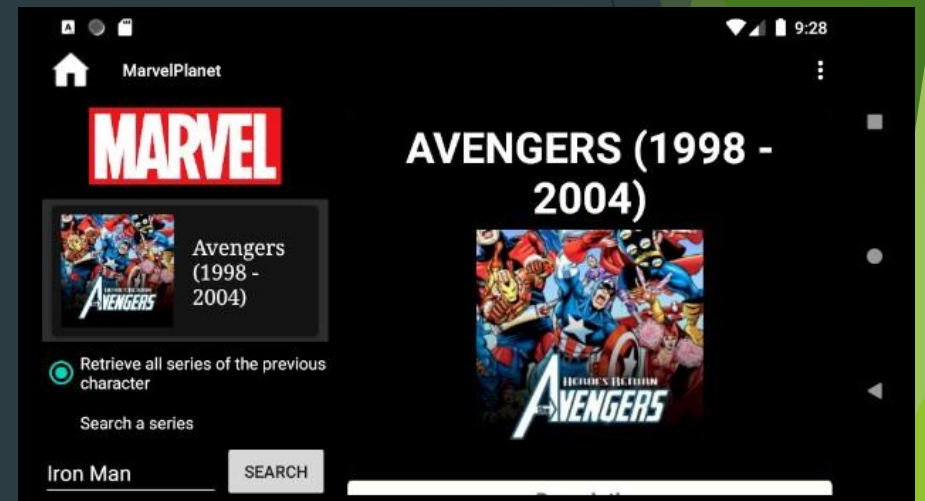
```
public void searchSeriesByTitleStartsWith(String s) {  
    String url = "https://gateway.marvel.com/v1/public/series?ts=%s&title=%s&apikey=%s&hash=%s";  
    String urlSearch = String.format(Locale.getDefault(), url, TIME_STAMP, s, APIKEY, HASH);  
    Log.w("TAG_M", s);  
    StringRequest stringRequest = new StringRequest(Request.Method.GET, urlSearch, this, error -> {});  
    requestQueue.add(stringRequest);  
}
```

Series

- Selezionando una serie si aprirà la pagina di dettaglio della serie

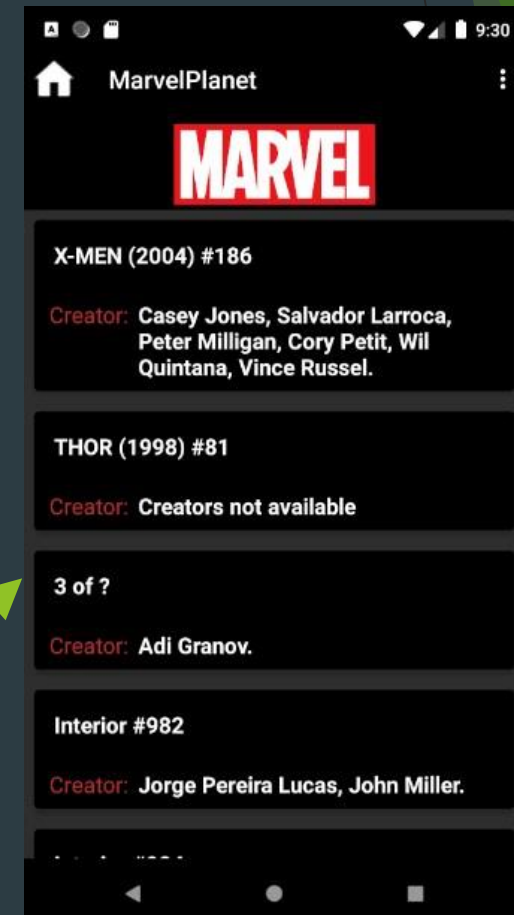
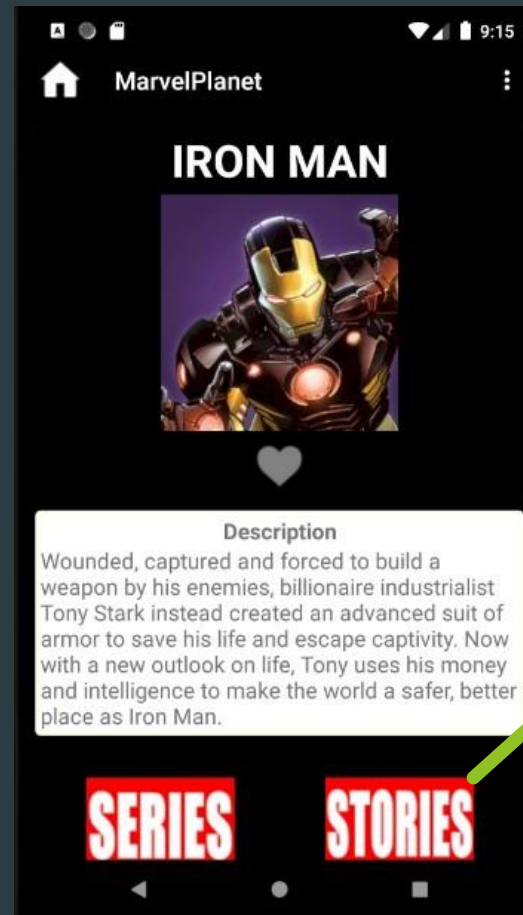


- Disponibile anche il *landscape* con lo stesso sistema di *fragment*



Stories

- Utilizzando il bottone **Stories** nella pagina di dettaglio dell'eroe, è possibile accedere alla sua lista di *storie*.
- Anche questa pagina è fatta da una *RecyclerView* che lo **StoriesAdapter** popola con *CardView* organizzate con:
 - *Nome della storia*
 - *Creatore della storia*



FRAGMENT DI DETTAGLIO

I detail's fragments, come suggerito dal nome, mostrano il contenuto dell'oggetto selezionato dall'utente nei list's fragment.

Dal punto di vista tecnico sono strutturati dichiarando la loro classe come estensione della libreria di supporto “*androidx.fragment.app.Fragment*”; in seguito viene fatto l'override dei metodi “*onCreateView*” e “*onViewCreated*”.

Nel primo metodo viene fatto l' "inflate" del layout di quel determinato fragment, ovvero sia viene creata la view.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_hero_detail, container, attachToRoot: false);
}
```

Nel secondo invece, che riceve come parametro una View e un Bundle, viene dichiarata una ViewModel alla quale vengono passati i parametri che confluiscono nella classe "Holder".

```
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    HeroViewModel hvm = new ViewModelProvider(requireActivity()).get(HeroViewModel.class);
    hvm.getInfo().observe(getViewLifecycleOwner(), item -> new Holder(view, item));
}
```

I ViewModel sono una classe necessaria alla gestione e il mantenimento dei dati da trasferire poi ad un'eventuale Activity o Fragment.

Essi fanno utilizzo dei LiveData che sono degli “observable” che acquisiscono rilievo in quanto sopravvivono al ciclo di vita.

```
public class SingleStoryViewModel extends ViewModel {  
    private MutableLiveData<Story> info;  
  
    public LiveData<Story> getInfo() {  
        if(info == null)  
            info = new MutableLiveData<>();  
        return info;  
    }  
  
    public void loadInfo(Story story) {  
        if(info == null)  
            info = new MutableLiveData<>();  
        info.setValue(story);  
    }  
}
```

Secondo i classici approcci, si dovrebbe ricorrere al salvataggio dello stato dell'applicazione tramite il metodo “onSaveInstanceState()”, ideale però solo per piccole quantità di dati.

Una struttura che basa invece il suo funzionamento sui ViewModel risulta utile quando la mole di dati transienti è cospicua e soprattutto fa in modo che non vengano penalizzate la consistenza dei dati e le prestazioni dell'applicazione.

L'ultimo elemento che confluisce nel codice dei fragment di dettaglio è rappresentato dall'Holder.

Esso altro non è che il tramite tra la stesura XML e quella Java.

Come è possibile notare dall'immagine infatti al suo interno vengono dichiarati le diverse tipologie di elementi che andranno a comporre la User Interface.

E' dotato di costruttore al quale come parametri vengono passati una view e un'entità ed al suo interno le varie Image e Text vengono collegate al loro corrispondente XML.

```
class Holder {  
    final TextView tvComicName;  
    final TextView tvComicDescription;  
    final TextView tvComicPrice;  
    final TextView tvComicPageNumber;  
    final ImageView ivComic;  
  
    Holder(View view, Comic comic) {  
        view.findViewById(R.id.svBase).setVisibility(View.VISIBLE);  
        tvComicName = view.findViewById(R.id.tvComicName);  
        tvComicDescription = view.findViewById(R.id.tvComicDescription);  
        tvComicPrice = view.findViewById(R.id.tvComicPrice);  
        tvComicPageNumber = view.findViewById(R.id.tvComicPageNumber);  
        ivComic = view.findViewById(R.id.ivComicRetr);  
  
        fillLayout(comic);  
    }  
}
```


Gli attributi dell'entità dichiarata nel costruttore, rendono possibile alla funzione “fillLayout” di impostare i valori delle varie TextView ed ImageView.

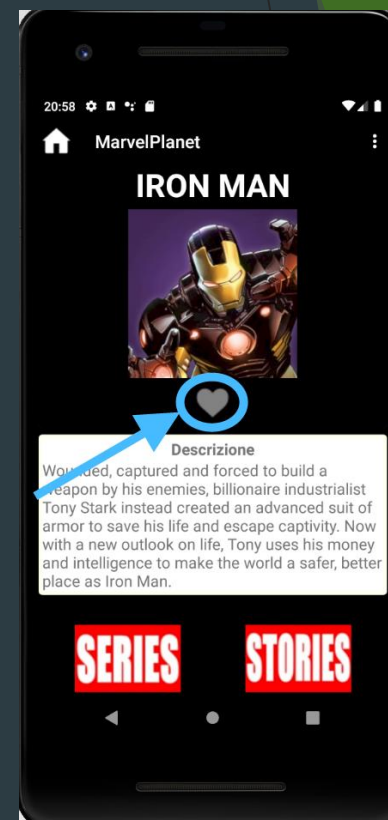
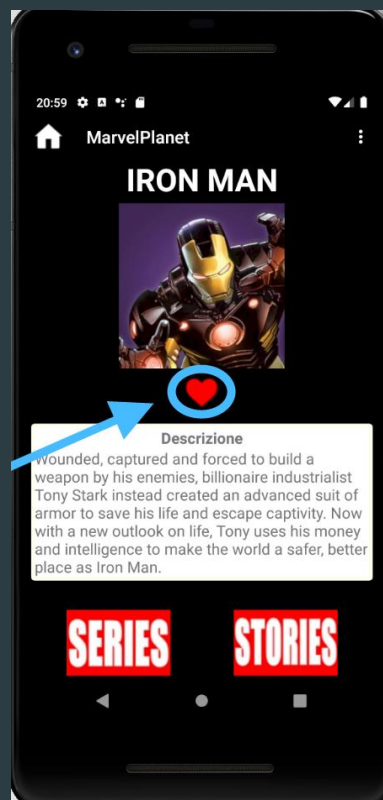
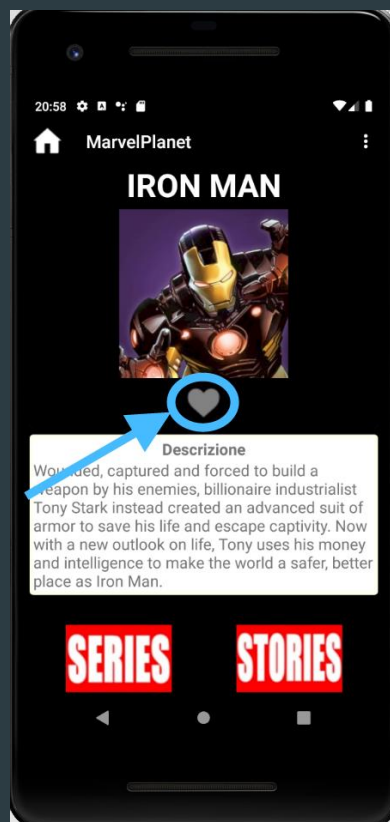
```
void fillLayout(Comic comic) {  
    tvComicName.setText(comic.getSeriesName().toUpperCase());  
    tvComicDescription.setText(comic.getDescription());  
    tvComicPrice.setText("10€");  
    tvComicPageNumber.setText(String.valueOf(comic.getPageCount()));  
  
    ImageRequest request = new ImageRequest(comic.getMarvelImage().getFullPath(),  
        ivComic::setImageBitmap, maxWidth: 0, maxHeight: 0, scaleType: null, Bitmap.Config.ARGB_8888,  
        error -> ivComic.setImageResource(R.drawable.iv_logo));  
    if(getActivity() != null) {...}  
}
```

PERSONAGGI PREFERITI

PERSONAGGI PREFERITI

- Per la realizzazione di questa funzionalità a livello grafico, è stata inserita una semplice icona a forma di cuore nella pagine di dettaglio del supereroe, che se cliccata permette di salvare il personaggio cercato in un'apposita sezione dell'applicazione in cui è possibile recuperare una lista di tutti gli eroi preferiti dell'utente. Tale icona, in questo punto dell'applicazione può essere selezionata, aggiungendo l'eroe nella lista dei preferiti, o anche deselezionata in un secondo momento generando la cancellazione del supereroe dai preferiti.

PERSONAGGI PREFERITI



primo click → aggiungo ai preferiti

Secondo click -> rimuovo dai preferiti

PERSONAGGI PREFERITI - ENTITY

- In questo punto dell'applicazione è presente un listener che si occupa di gestire il click sull'icona del cuore. Il listener, se l'icona del cuore NON è selezionata va a creare una nuova istanza dell'entità *FavouriteHero* e a salvarla sul database dopo aver valorizzato tutti gli attributi del *FavouriteHero* da salvare. Invece, se l'icona del cuore è selezionata, e viene intercettato un nuovo click su di essa allora il listener si occuperà di andare ad eliminare tale supereroe dai preferiti.

```
@Entity(tableName="heroes")
public class FavouriteHero implements Parcelable {

    @ColumnInfo(name="_id")
    public int id;
    @PrimaryKey@NonNull
    @ColumnInfo(name="name")
    public String name;
    @ColumnInfo(name="description")
    public String description;
    @ColumnInfo(name="resURI")
    public String resURI;
    @ColumnInfo(name="imgHero")
    public String imgHero;
    @ColumnInfo(name="collectionURIComics")
    public String collectionURIComics;
    @ColumnInfo(name="collectionURISeries")
    public String collectionURISeries;
    @ColumnInfo(name="collectionURISTories")
    public String collectionURISTories;
    @ColumnInfo(name="collectionURIEvents")
    public String collectionURIEvents;
    @ColumnInfo(name="pref_hero")
    public int pref_hero = 0;
```

PERSONAGGI PREFERITI - ROOM DATABASE

Il database è stato creato usando *Room* creando una classe astratta che estende *RoomDatabase*, includendo l'entità *FavouriteHero* all'interno dell'annotazione ed includendo un metodo astratto che restituisce la classe che è annotata con *@Dao*

```
@Database(entities={FavouriteHero.class}, version = 1)
public abstract class MyAppDatabase extends RoomDatabase {

    public static MyAppDatabase instance;
    public abstract MyDAO myDAO();

    public static synchronized MyAppDatabase getInstance(Context context){
        if(instance == null){
            instance =Room.databaseBuilder(context.getApplicationContext(),
                MyAppDatabase.class, name: "marvelldb")
                .fallbackToDestructiveMigration()
                .build();
        }
        return instance;
    }
}
```

PERSONAGGI PREFERITI - DAO

La classe Dao contiene diversi metodi per accedere al database, tra cui:

- **addHero** (per aggiunge un eroe ai preferiti)
- **deleteHeroFromPrefs** (per eliminare un eroe dalla lista dei preferiti)
- **getPref** (per verificare se l'eroe è presente nella lista dei preferiti)

```
@Dao
public interface MyDAO {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void addHero(FavouriteHero hero);

    @Query("UPDATE heroes SET pref_hero = :value WHERE name = :name")
    void update(int value, String name);

    @Delete
    void deleteHeroFromPrefs(FavouriteHero name);

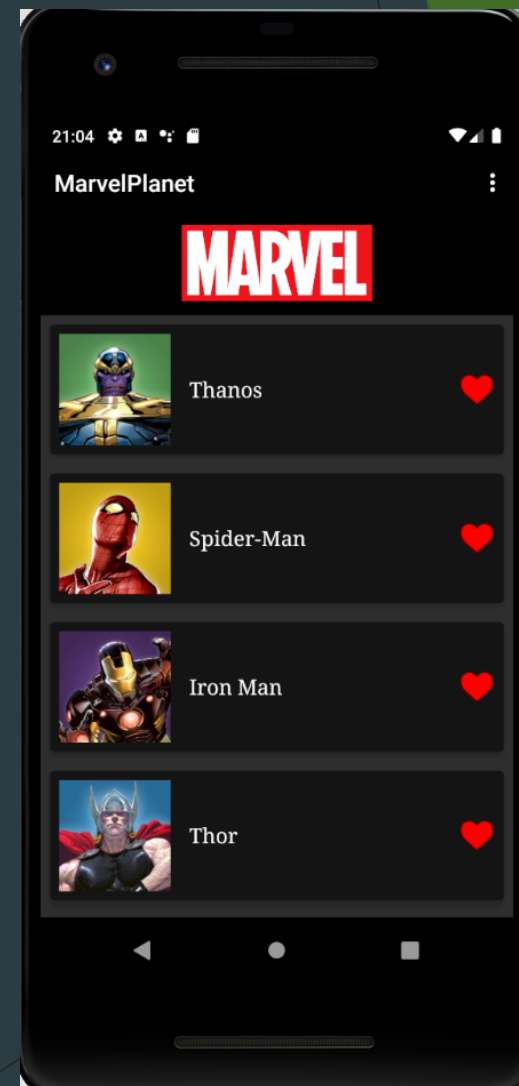
    @Query("select pref_hero from heroes where name = :name")
    int getPref(String name);

    @Query("select * from heroes where name = :name")
    public FavouriteHero getHero(String name);

    @Query("select * from heroes")
    public List<FavouriteHero> getHeroes();
}
```

PERSONAGGI PREFERITI - LISTA

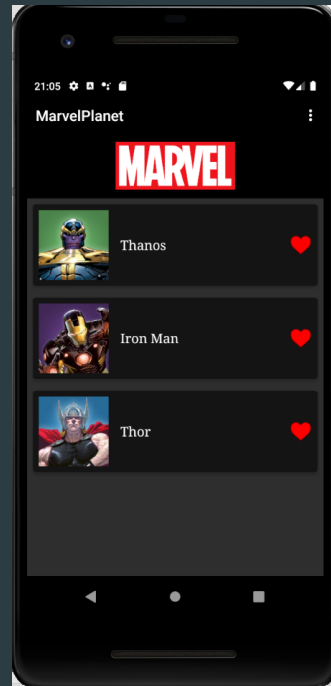
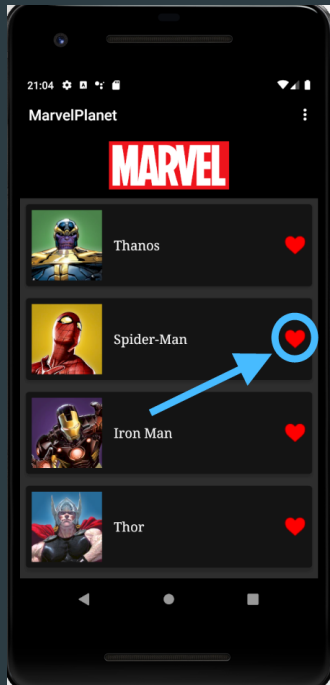
- Una volta aggiunto un eroe ai preferiti cliccando sull'icona del cuore nella pagina di dettaglio del supereroe, è possibile accedere attraverso la sezione preferiti del menu principale ad una lista in cui sono presenti tutti i preferiti dell'utente.



PERSONAGGI PREFERITI - LISTA

- ▶ Su ogni singolo eroe preferito che compone la lista è possibile:
 - ▶ Rimuovere il personaggio dai preferiti direttamente dalla lista, cliccando sull'icona del cuore.
 - ▶ Accedere alla pagina di dettaglio del supereroe preferito cliccando su quest'ultimo.

PERSONAGGI PREFERITI - LISTA



Rimozione dai preferiti dell'eroe

Accesso alla pagina di dettaglio del supereroe preferito

