UNIVERSITÀ DEGLI STUDI DI CAMERINO

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea di Informatica Classe L-31

STUDIO E SVILUPPO IN LINGUAGGIO KOTLIN DI UN'APPLICAZIONE MOBILE ANDROID

Tesi di Laurea di:

Angelo Sercecchi

Relatore:

Prof. Fausto Marcantoni

Correlatore:

Dott. Daniele Benedetti

Anno accademico 2019/2020

Abstract

La tesi di Laurea descrive il lavoro svolto presso l'azienda "Be Ready Software" di Corridonia per lo sviluppo di un'applicazione mobile Android. Questa collaborazione è nata dopo uno stage con la medesima impresa, richiedendo una quantità di lavoro pari a trecento ore. Grazie ai buoni rapporti lavorativi e professionali, creatisi durante tale esperienza, è stato possibile sviluppare questo progetto e redigere questa tesi di Laurea. L'obiettivo della tesi è quello di presentare l'applicazione sviluppata trattando gli aspetti teorici, grafici e del linguaggio di programmazione utilizzato. Viene proposta un'introduzione all'ambiente di sviluppo *Android Studio*, un approfondimento del linguaggio di programmazione *Kotlin* e una spiegazione e anteprima delle varie funzionalità e aspetti dell'applicazione finale. Nello specifico l'applicazione è stata sviluppata su richiesta di ADI Museum Compasso d'Oro, il quale raccoglie a Milano gli oltre 350 pezzi premiati con il premio Compasso d'Oro. Tramite questa app è possibile visualizzare le varie opere, ascoltare le rispettive audioguide, orientarsi tramite una mappa e, con una implementazione futura, acquistare ticket online.

STUDIO E SVILUPPO IN LINGUAGGIO KOTLIN DI UN'APPLICAZIONE MOBILE ANDROID

Angelo Sercecchi

Indice

Introduzione

L'azienda *Android Inc.*, fondata nel 2003 da Andy Rubin, Rich Miner, Nick Sears e Chris White, aveva come scopo quello di sviluppare "dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario". È solo nel 2005, con l'acquisizione da parte di Google, che Android inizia a diventare un sistema operativo per dispositivi mobili, basato su un kernel Linux. Ci vollero altri due anni di sviluppo, da parte del team di Rubin, prima della presentazione ufficiale avvenuta il 5 Novembre 2007. Da allora i dispositivi mobile con sistema operativo Android sono aumentati ogni anno.

Le applicazioni, generalmente, sono sviluppate in linguaggio *Java* utilizzando il kit di sviluppo software Android (SDK). Dal 7 maggio 2019, tuttavia, Google ha reso *Kotlin* il linguaggio ufficiale per lo sviluppo in Android. Kotlin è un linguaggio di programmazione general purpose e multi-paradigma basato sulla JVM (Java Virtual Machine) che riporta elementi mutuati da Java e Scala con diversi aspetti sintattici derivati direttamente da Pascal e Go. Nel dettaglio, è un linguaggio a tipizzazione statica e forte, orientato verso la programmazione ad oggetti permettendo anche un pieno uso dell'approccio funzionale. È totalmente compatibile con JDK 6, fornendo delle prestazioni pari e in alcuni casi superiori a Java. Viene garantita l'interoperabilità con Java al 100%. Il passaggio da Java a Kotlin per un programmatore è reso molto semplice per l'affinità dei due linguaggi e dagli strumenti di conversione resi disponibili. Si è pertanto deciso, attraverso questa tesi di Laurea, di approfondire lo studio del nuovo linguaggio di programmazione concludendo con lo sviluppo di un'applicazione Android.

Le basi dell'interfaccia grafica

In un software, qualunque esso sia, per poter interagire con l'utente sia in input che in output è fondamentale l'interfaccia grafica. Una sua conoscenza approfondita, inoltre, permette di rendere l'aspetto grafico accattivante e funzionale alle esigenze dell'utente finale.

Nello specifico, la classe fondamentale dell'interfaccia Android è *View*. Questa classe viene estesa da qualunque elemento che appare a livello grafico e che può svolgere una o entrambe di queste funzionalità:

- Mostrare un aspetto.
- Gestire eventi di interazione con l'utente.

Ci sono principalmente tre categorie di view:

- *Layout*: è un contenitore che ha il compito di definire una struttura visiva dei suoi figli. (*es: LinearLayout, RelativeLayout, ConstraintLayout*)
- *Controlli utente*: sono delle view che permettono un'interazione con l'utente. (*es: TextView, Button, EditText, Spinner, ImageView*)
- *AdapterView***:** vengono utilizzati per generare in modo ciclico grafiche responsive. (*es: ListView, GridView*)

Qualsiasi elemento grafico eredita, dalla sua classe padre View, l'attributo id. Questo identificativo è utile per etichettare l'elemento della UI e per potervi fare riferimento nel codice di programmazione Kotlin o in altre risorse XML. L'identificativo deve essere univoco all'interno dello stesso file xml.

```
<TextView>
    android:id="@+id/nome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    ...
</TextView>
```

Si potrà fare riferimento al precedente elemento grafico tramite l'utilizzo del suo id:

- Nel file *xml*: tramite "@id/nome"
- Nel codice *Kotlin*: tramite R.id.nome

I layout

Nel framework Android sono stati definiti vari tipi di layout di diverso funzionamento grafico per facilitare la gestione di situazioni e contesti differenti.

LinearLayout

Questo tipo di layout contiene un insieme di elementi che vengono distribuiti in maniera sequenziale dall'alto verso il basso (se definito con orientamento verticale) o da sinistra a destra (se ha un orientamento orizzontale). È un layout molto semplice e piuttosto naturale per i display di smartphone e tablet;

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    ...
    ...

</LinearLayout>
```

TableLayout

Questo layout inquadra gli elementi in una tabella, quindi è particolarmente adatto a mostrare strutture regolari suddivise in righe e colonne come form o griglie. È piuttosto semplice da usare e ricorda molto le tabelle HTML nelle pagine web con i ben noti tag <table> <tr> <td>;

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*" >

      <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
            ...
            ...
      </TableRow>

      ...
      ...

</TableLayout>
```

RelativeLayout

Sicuramente è uno dei layout più flessibili e moderni ed è adatto a disporre in maniera meno strutturata gli elementi. Questi, essendo *relative*, si posizionano in relazione l'uno all'altro o rispetto al loro contenitore, permettendo un layout fluido che si adatta bene a display diversi. Rispetto ai due precedenti è ricco di attributi XML che servono ad allineare e posizionare gli elementi tra loro;

```
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    ...
    ...

</RelativeLayout>
```

ConstraintLayout

E' il layout più evoluto, nato appositamente per la creazione di interfacce responsive. Consente di creare layout ampi e complessi con una gerarchia semplice di elementi grafici. Ha diverse similitudini con il RelativeLayout ma è più flessibile e facile da usare con l'Editor Layout di Android Studio.

```
<android.support.constraint.ConstraintLayout
    android:layout_width="255dp"
    android:layout_height="70dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"/>

    ...
    ...

</android.support.constraint.ConstraintLayout>
```

Controlli utente

Gli elementi grafici facenti parte di questa tipologia gestiscono un'interazione con l'utente permettendogli sia l'inserimento di dati (input) sia una loro visione (output).

TextView

Questa view è una delle fondamenta dell'interfaccia grafica e si trova in qualsiasi applicazione Android. Questo elemento permette la visualizzazione di testo. Tramite questo componente è possibile formattare il testo al suo interno mediante attributi standard come *textStyle*, *textSize*, *fontFamily* e *color*.

```
<TextView
    android:text="testo da scrivere"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="36sp"
    android:fontFamily="sans-serif-light"
    android:textColor="@android:color/black"/>
```

Button

Permette l'interazione con l'utente tramite un click. Un bottone può essere personalizzato con uno stile definendo il testo, una sua possibile icona, forma o colore. Al tocco su di esso si scaturisce un evento il quale può essere collegato ad un funzione.

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="testo button"
    android:drawableLeft="@android:drawable/icona"
    android:background="@android:color/red"
    android:onClick="funzioneDiCollegamento" />
```

EditText

Questo elemento dell'interfaccia grafica viene utilizzato per l'immissione e la modifica del testo da parte dell'utente. È possibile definire il tipo di valori in input

tramite l'attributo xml *inputType*. Di default questo viene settato con "text" ma può assumere stati diversi come "number", "decimalNumber", "email", "phone" e molti altri. In base al tipo di input definito si aprirà, in automatico, la tastiera corrispondente.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:hint="@string/hello_world"
    android:singleLine="true"
    android:inputType="numberDecimal"
    android:digits="0123456789."
    android:maxLength="6"/>
```

Spinner

Fornisce un modo rapido per selezionare un valore da una lista. Nello stato predefinito, una casella di selezione mostra il valore attualmente selezionato. Toccando la casella di selezione viene visualizzato un menù a tendina con tutti i valori disponibili, da cui l'utente può selezionarne uno nuovo.

```
<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:popupBackground="@android:color/white"
    android:scrollbars="none"
    android:spinnerMode="dropdown" />
```

ImageView

Questo elemento grafico permette la visualizzazione di un'immagine indicata tramite l'attributo *src*. Rende possibile il suo ridimensionamento, l'applicazione di tinte ed un suo eventuale zoom.

```
<ImageView
    android:layout_width="20sp"
    android:layout_height="20sp"
    android:tint="@android:color/yellow"
    android:src="@android:mipmap/ic_launcher"/>
```

AdapterView

Questi componenti grafici sono collegati ad un adapter e raccolgono tutte le View prodotte da esso per mostrarle secondo il proprio standard. Gli AdapterView si limitano in molti casi a recuperare la View dall'adapter e a mostrarle nel layout. L'adapter di cui stiamo parlando è un componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi, per ogni elemento dell'AdapterView. In *Kotlin*, per impostare l'adapter in un AdapterView si utilizza il metodo *setAdapter()* su quest'ultimo.

ListView

Questo elemento grafico inserisce le view generate dall'adapter in verticale. Ognuna di essa occuperà una riga generando così una lista scrollabile.

```
<ListView
    android:id="@+id/listviews"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

GridView

Permette di visualizzare le differenti view dell'adapter tramite una tabella. Si possono definire il numero di colonne della tabella tramite l'attributo xml *numColumns***.**

```
<GridView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:numColumns="2"
    android:columnWidth="90dp"
    android:horizontalSpacing="10dp"
    android:verticalSpacing="10dp"
    android:gravity="center"/>
```

Activity

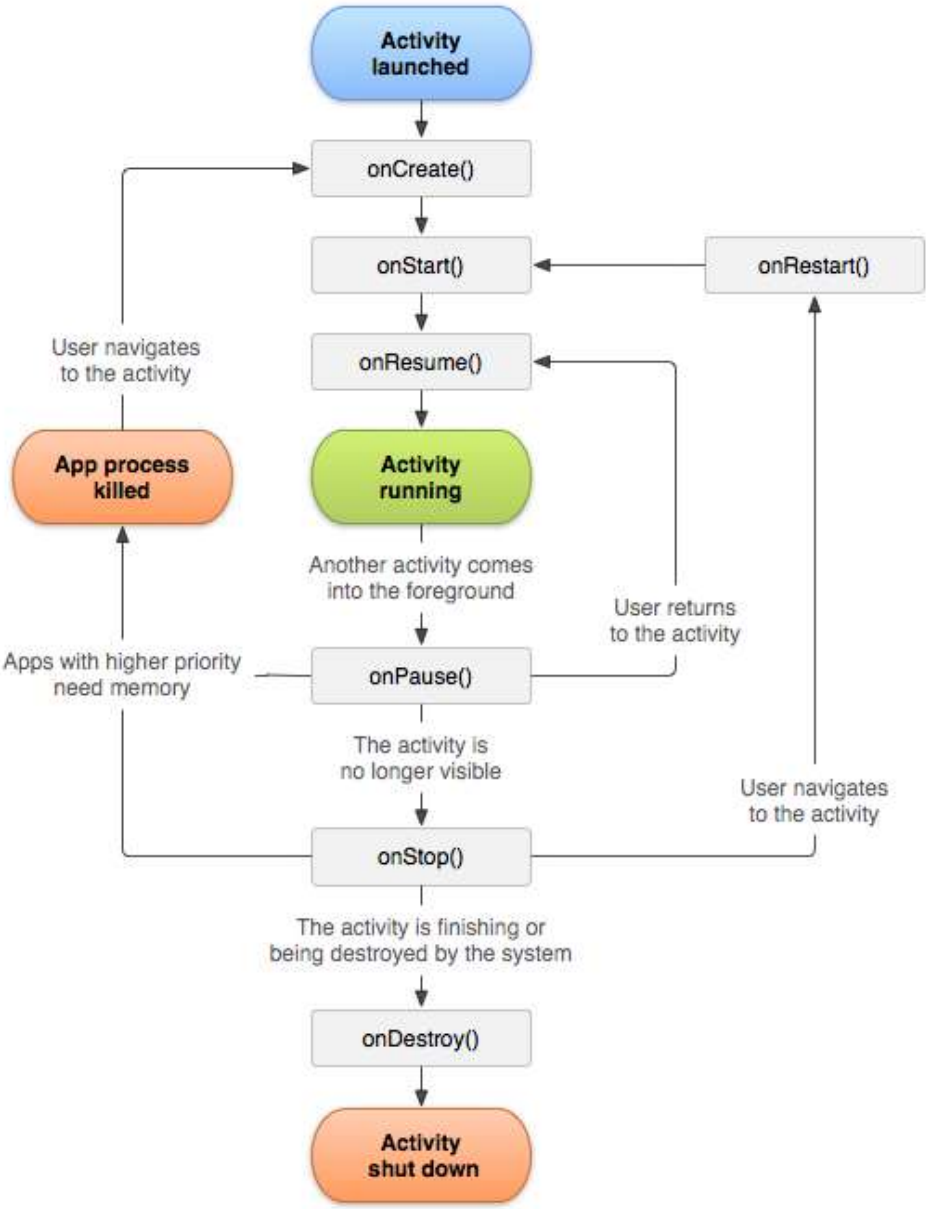
Un’Activity è il punto di ingresso per l'interazione con l'utente, essa rappresenta una singola schermata dell’interfaccia. Sebbene le Activity lavorino insieme per formare un'esperienza utente coerente nell'applicazione che si sviluppa, ognuna di esse è indipendente dalle altre. Questo componente facilita le interazioni chiave tra app e sistema in quanto:

- Tiene traccia di ciò che attualmente interessa all'utente (ciò che è sullo schermo) per garantire che il sistema continui ad eseguire il processo che ospita l'attività.
- Traccia i processi precedentemente utilizzati (activity interrotte), nell’eventualità che l’utente voglia riutilizzare l’activity.
- Aiuta l'app a gestire l'arresto del processo in modo che l'utente possa tornare alle attività con il ripristino dello stato precedente.

Un’activity può essere vista come la rappresentazione di una schermata (screen). La piattaforma Android mantiene in primo piano (foreground) un solo screen alla volta, mentre i rimanenti vengono messi in secondo piano (background). Il tasto back permette di passare da uno screen all'altro, portando in background lo screen attuale e assegnando il foreground al precedente. Questa *history* viene chiamata *activity stack*, gestita dall'Activity Manager.

Ciclo di vita di un Activity

Il ciclo di vita di un’activity rappresenta una catena di stati che essa attraversa lungo la sua durata. Gran parte di questi stati sono indotti dal sistema operativo, il quale per un corretto utilizzo di risorse gestisce le activity in base alle operazioni svolte dall’utente.



Nell'immagine sovrastante, tramite le forme colorate si distinguono le fasi più importanti della vita di un'activity: il suo avvio, la fase di running ed infine la sua rimozione. I rettangoli grigi rappresentano dei metodi di callback, esposti dalla classe Activity, che il sistema chiamerà man mano che il ciclo di vita di questa componente progredisce. E' molto importante capire l'utilizzo di ogni singolo evento per poter sviluppare un'applicazione di buona qualità. Quando un'activity va in esecuzione per interagire direttamente con l'utente vengono obbligatoriamente invocati tre metodi:

- *onCreate*: è richiamato quando l'activity viene avviata per la prima volta. In questo metodo si devono assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia;

- *onStart*: l'activity sta per essere visualizzata. È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente;
- *onResume*: l'activity diventa la destinataria di tutti gli input dell'utente ed inizia ad interagire con esso.

Quando l'utente chiude o mette in background un'attività del sistema, Android pone a riposo l'activity. Tale evento potrebbe verificarsi, per esempio, quando viene aperta un'altra applicazione, si riceve una chiamata oppure, più semplicemente, si apre un'altra activity della medesima app. Questo percorso passa per tre metodi di callback:

- *onPause***:** richiamato quando l'activity sta per andare in background, solitamente in quanto è stata avviata un'altra activity che si prepara a prendere il foreground.
- *onStop***:** viene invocato quando l'activity non è più visibile all'utente. Nel caso in cui la memoria fosse poca, allora, Android terminerà il processo associato a tale applicazione.
- *onDestroy***:** viene richiamato appena prima della distruzione dell'activity.

Bisogna tenere a mente che i metodi di callback sono stati concepiti a coppie: un metodo di avvio con un corrispondente metodo di arresto (*onCreate–onDestroy*, *onStart–onStop*, *onResume–onPause).* Le funzionalità e le risorse allocate nel metodo di avvio, solitamente, verranno annullate nel relativo metodo di arresto.

Sono presenti altri due metodi di callback, meno utilizzati, ma che meritano ugualmente un approfondimento:

- *onSaveInstanceState*: questo metodo viene invocato da Android per salvare alcune informazioni di stato dell'activity come, per esempio, la posizione del cursore all'interno di una editText. Normalmente non si eseguono operazioni in questa funzione in quanto Android salva per noi queste informazioni di stato.

- *onRestoreInstanceState***:** viene richiamato solo nel caso in cui alcuni stati dell'activity sono precedentemente stati salvati con onSaveInstanceState.

Fragment

Un fragment rappresenta una parte riutilizzabile dell'interfaccia utente dell'applicazione. Può essere visto, per l'appunto, come un frammento di un'activity, con un proprio ciclo di vita e con la possibilità di definire un determinato layout. Un fragment non può "vivere" da solo: ha bisogno di un activity ospitante. Questo elemento costituisce uno dei più importanti mezzi per lo sviluppo di interfacce grafiche moderne. Dopo il suo inserimento, tramite la versione Android 3.0, la sua conoscenza è diventata cardine per ogni Android Developer.

Ciclo di vita di un Fragment

Il ciclo di vita di un Fragment è fortemente collegato alla sua relativa activity di appartenenza. Molti metodi di callback sono gli equivalenti dell'activity, i più utilizzati sono:

- *onAttach*: questo metodo segnala il momento in cui il Fragment scopre il rispettivo activity di appartenenza. Solitamente, in questo punto l'activity non è stata ancora creata per cui non si può interagire con essa.
- *onCreate*: rappresenta la creazione del Fragment, non sotto l'aspetto grafico ma in quanto componente.
- *onCreateView*: segnala il momento in cui viene creato il layout. In questo metodo si definisce l'interfaccia utente relativa al fragment.
- *onActivityCreated*: questo metodo indica il completamento della creazione dell'activity d'appartenenza.

I metodi rimanenti del ciclo di vita vengono chiamati in corrispondenza degli omonimi metodi dell'activity rispettiva (*onStart*, *onResume*, *onPause*, *onStop*, *onDestroyView*, *onDestroy*, *onDetach*).

Kotlin

Kotlin è un linguaggio di programmazione sintetico, potente, altamente espressivo e dominato da una mentalità moderna. Questo linguaggio deriva da Java e può essere usato in sua alternativa. È strutturato per interoperare con la piattaforma Java Runtime Environment come target principale. Questo garantisce il funzionamento delle applicazioni in ogni ambiente che accetti la JVM. Per un programmatore, il passaggio da Java a Kotlin è alquanto facile grazie all'affinità dei due linguaggi di programmazione e agli strumenti di conversione reperibili. Il 7 maggio 2019 Google ha reso *Kotlin* il linguaggio ufficiale per lo sviluppo in Android.

Kotlin è stato progettato con il preciso obiettivo di essere un linguaggio moderno. Alcune delle caratteristiche che lo rendono tale sono:

- Sintassi pulita che permette una lettura ed una scrittura del codice più fluida.
- Operator Overloading che consente la creazione di una famiglia di funzioni aventi tutte lo stesso nome, ma con signature diversa ed eventualmente anche diverso valore di ritorno.
- Type-Inference serve per intuire il tipo di dato più adatto per il valore passato in inizializzazione. Questo processo velocizza molto la scrittura di codice.

- Null-Safety: il linguaggio implementa una serie di politiche che cercano di evitare il verificarsi delle eccezioni NullPointerException. Qualsiasi riferimento può essere nullable o non-nullable.

Variabili e constanti

Una variabile mutabile, in Kotlin, deve essere dichiarata tramite l'utilizzo della parola chiave *var*. Invece, per dichiarare una variabile immutabile (una costante) bisogna utilizzare la parola *val*. L'inizializzazione di una variabile con un valore *null* può essere possibile solo se il suo tipo è seguito dal carattere chiave *?.*

```
val voto : Integer = 2
var nome : String? = null
```

In questo esempio la costante "voto" rimarrà per sempre immutata ed uguale a 2, mentre la variabile "nome" può assumere il valore *null* e il suo contenuto può essere riassegnato.

Costrutti condizionali

I costrutti condizionali sono delle espressioni che consentono di eseguire una certa porzione di codice invece di un'altra, in base ad una "scelta". In questo linguaggio di programmazione vengono principalmente usati tre costrutti: *if*, *if-else*, *when*.

Condizione if

Questa condizione rappresenta l'istruzione più semplice ed elementare. La condizione *if* formulata in una frase, fondamentalmente, dice: "Solo se questa condizione è vera, esegui questa porzione di codice". Nel caso in cui la condizione non fosse vera, allora il programma salterà semplicemente quel blocco di codice.

Condizione if-else

L'istruzione *if-else* può essere definita come "evoluzione" della precedente. L'unica differenza è che, se la condizione booleana non è soddisfatta, verrà eseguito il blocco di codice all'interno dell'*else*.

```
val votoFinale=8

if (votoFinale>=6){
     println("Sei stato PROMOSSO")
}else{
      println("Purtroppo ti hanno BOCCIATO")
}
```

Nell'esempio soprastante, verrà stampata la scritta "Sei stato PROMOSSO" in quanto *votoFinale* è maggiore od uguale a 6.

Costrutto when

Questo costrutto prende il posto del più comune *switch-case* con una sintassi più moderna e leggibile. La parola chiave *when* riceve tra parentesi tonde una variabile da valutare. Nel blocco di codice che segue vengono elencati i suoi possibili valori e il relativo codice da eseguire. C'è anche la possibilità, tramite la parola chiave *else*, di definire un caso di *default* che verrà eseguito solo quando la variabile non corrisponde a nessun valore precedente.

```
val voto=3
val risultato=when(voto)
{
   in 0..4 -> "Gravemente insufficiente"
   5 -> "Insufficiente"
   in 6..8 -> "Buono"
   9, 10 -> "Ottimo"
   else -> "Valore non valido"
}
println(risultato)
```

In questo esempio si avrà come risultato "Gravemente insufficiente" per voti compresi tra 0 e 4, "Insufficiente" per il 5, "Buono" per voti compresi tra 6 e 8, "Ottimo" per il 9 e 10 mentre per qualsiasi altro valore verrà restituito il messaggio "Valore non valido". Nello specifico dell'esempio viene stampato il

testo “Gravemente insufficiente” in quanto la variabile analizzata, cioè voto, equivale a 3.

Cicli

Nella programmazione molte volte è necessario svolgere un blocco di codice in modo iterativo e questa operazione viene svolta tramite i loop. Kotlin offre diverse strutture per creare un ciclo, ognuna con caratteristiche differenti.

Ciclo for

Questa tipologia di ciclo viene utilizzata per ripetere qualcosa per un determinato numero di volte. Un *ciclo for* esegue una serie di istruzioni per ogni elemento di un intervallo, sequenza o collection.

```
for (x in 1..10) {
    println(x)
}
```

Nell’esempio precedente verranno ciclati tutti i numeri compresi nell’intervallo da 1 a 10, tale valore verrà inserito nella variabile “x” e questa verrà stampata. L’output di questo blocco di codice sarà quindi*: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10*.

Ciclo while

Questa tipologia di ciclo continuerà a ripetersi finché la condizione booleana specificata non sarà verificata (quindi false). Con il normale *while* la condizione viene controllata all’inizio di ogni ciclo, questo significa che, nell’eventualità che questa sia subito falsa, il ciclo potrebbe essere saltato completamente. Il blocco di codice all’interno del ciclo dovrà eseguire un determinato lavoro per far sì che la condizione risulti falsa, se questo non viene fatto si potrebbe entrare in un loop infinito.

```
var contatore=1
while (contatore<10)
{
   println(contatore)
   contatore++
}
```

Un'alternativa al *while* standard è il *do-while* dove la condizione viene verificata alla fine di ogni iterazione. Questo permetterà di eseguire il blocco di istruzioni all'interno del ciclo almeno una volta.

Break e continue

Due parole chiavi all'interno di qualsiasi struttura ciclica sono *break* e *continue*. Il primo permette di interrompere ed uscire dal ciclo immediatamente, il secondo permette di interrompere l'iterazione corrente passando direttamente a quella successiva.

Funzioni

Le funzioni sono porzioni di codice con una *signature* univoca che svolgono compiti precisi. Queste possono essere invocate più volte e su parti diverse dell'applicazione. Tramite le funzioni è possibile rendere il codice più pulito e fluido. Una funzione può anche essere aparametrica, cioè che non accetta alcun parametro, oppure può avere differenti parametri e un valore di ritorno.

```
fun sum(x: Int, y: Int): Int {
    return x + y
}
```

La funzione d'esempio prende in input due valori interi (*x* e *y)*, li somma e li restituisce tramite la *keyword return*.

Classi e oggetti

In Kotlin, si applicano tutti i paradigmi tradizionali della programmazione orientata agli oggetti con una serie di innovazioni che rendono il linguaggio estremamente più moderno.

Per creare una classe, si deve utilizzare la parola chiave *class*. Questa *keyword* è seguita dal nome della classe

```
class Alunno(nomeAlunno:String, cognomeAlunno:String){
    var nome=nomeAlunno
    var cognome=cognomeAlunno
}
```

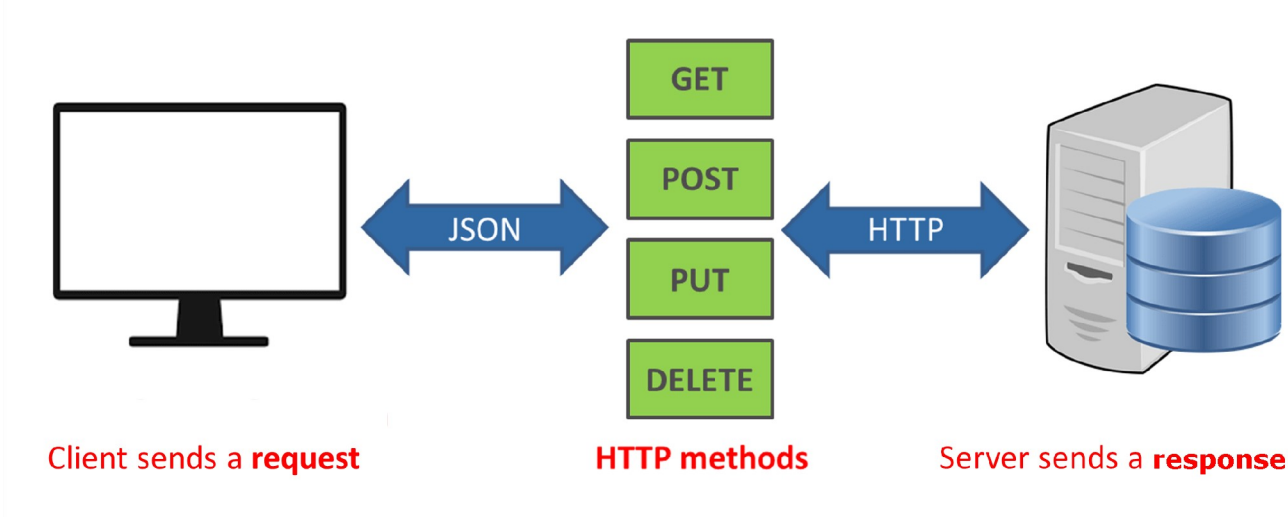
Da notare che, dopo la definizione (*class Alunno*), vengono dichiarate alcune variabili (*nomeAlunno* e *cognomeAlunno*). In questo modo si indica il metodo costruttore in Kotlin. Le variabili in esso nominate vengono direttamente utilizzate per inizializzare le variabili di istanza. Il limite di questo tipo di costruttore è rappresentato dall'impossibilità di far intervenire del codice durante l'assegnazione dei valori. Ciò può essere risolto con un blocco *init* che interviene in fase di inizializzazione. Nel caso in cui le variabili di istanza vengano dichiarate con *val*, cioè come costanti, allora queste saranno solo leggibili e non modificabili (proprietà *read-only*). Per effettuare l'overloading sui costruttori si utilizza la parola chiave *constructor* con il relativo metodo (simile a Java).

```
class Alunno(nomeAlunno:String){
    var nome=nomeAlunno
    var cognome : String? =null

    constructor(nomeAlunno:String, cognomeAlunno: String): this(nomeAlunno){
        cognome=cognomeAlunno
    }
}
```

API

Con il termine API (Application Programming Interface) si intende un set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi. Consentono ai propri prodotti o servizi di comunicare con altri prodotti o servizi senza conoscere la loro implementazione. In questo modo si semplifica lo sviluppo dell'app consentendo un netto risparmio in termini di tempo e denaro. La loro finalità è quella di ottenere un'astrazione a più alto livello, solitamente tra l'hardware e il programmatore o tra software a basso e ad alto livello, semplificando così il lavoro di programmazione. Le API, con facilità, rendono possibile collegare l'infrastruttura tramite lo sviluppo di app cloud-native e condividere dati con i clienti o con altri enti esterni.



Metodi http

L'HTTP è il protocollo che permette di inviare e ricevere documenti attraverso il web. Un protocollo non è altro che un insieme di regole che determina quali messaggi possono essere scambiati e quali rappresentano risposte appropriate. Un protocollo noto e molto utilizzato è, per esempio, POP3 il quale permette di ricevere le email.

I messaggi http sono composti da un'intestazione (header) e da un corpo (body), quest'ultimo spesso può rimanere vuoto. Solitamente nel body si inseriscono le

informazioni che si vogliono trasmettere tramite la rete. L'header, invece, contiene metadati, per esempio le informazioni di codifica. Ogni API o risorsa è identificata con un URL ben specifico. Se si vuole fruire di tale risorsa bisogna utilizzare il relativo URL. Esistono principalmente cinque tipologie di metodi HTTP:

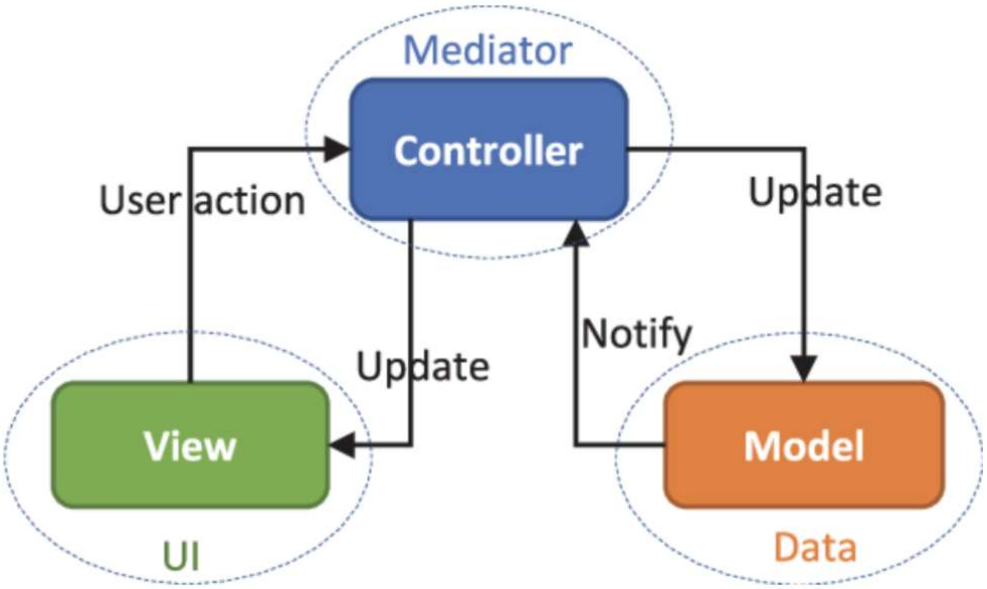
- *GET*: rappresenta il tipo di metodo di richiesta http più semplice. Tramite questa tipologia si richiede al server di trasmettere le informazioni identificate tramite quell'URL. Per cui, una richiesta GET è di sola lettura, un modo per prendere delle informazioni da un server.
- *POST*: questo tipo di richiesta, viene utilizzata per mandare delle informazioni al server tramite il campo body. Possiamo identificare questo tipo di metodo come di scrittura. Potrebbe essere un modo per salvare o modificare informazioni nel server.
- *DELETE*: tramite questa tipologia di metodo è possibile cancellare la risorsa specificata nell'URL della richiesta.
- *PUT*: come tipo di richiesta è molto similare alla POST. Infatti, permette di modificare o aggiungere risorse nel server. L'unica differenza è che bisogna identificare la specifica risorsa da aggiornare tramite l'URL.

Pattern architetturale - MVC

Un pattern architetturale può essere definito come una soluzione generale e riutilizzabile per un problema comune presente all'interno di un particolare contesto. Un pattern descrive lo scheletro principale che dovrà avere il software, i suoi servizi e come questi dovranno lavorare, le parti coinvolte e la logica di interazione tra di esse.

Esistono molteplici standard per questa tipologia di pattern ma, senza dubbio, il più utilizzato nella programmazione ad oggetti è l'MVC pattern (Model-View-Controller). Questo tipo di architettura permette di separare la rappresentazione interna dello stato dell'applicazione (Model), le informazioni ed elementi grafici della vista (View) e la logica che controlla l'interazione con lo stato dell'applicazione (Controller).

Lo schema sottostante fornisce una visione del funzionamento delle interazioni di ogni livello con i rimanenti. Ogni parte di questa architettura ha un ruolo e delle regole ben precise per la comunicazione con le altre parti. Questo permette allo sviluppatore del software di poter aggiornare o sostituire un livello della tecnologia senza dover modificare altri livelli.



Vediamo nello specifico i diversi livelli e i loro ruoli:

- *Model*: questa componente ha la responsabilità di rappresentare lo stato della nostra applicazione e di fornire metodi per la loro reperibilità e modifica. Una model può essere vista semplicemente come una classe che rappresenta un tipo di dato utilizzato nell'applicazione (come nell'esempio precedente con la classe *Alunno*).
- *View*: questo livello ha come scopo principale quello di creare un'interfaccia grafica per la visualizzazione dei dati (model) ed un'interazione con l'utente, il quale potrebbe svolgere azioni per modificare lo stato attuale dei dati o dell'applicazione in generale.
- *Controller*: ha il compito di fungere da ponte tra le due componenti precedenti. Riceve i diversi comandi generati dell'utente attraverso l'interazione con la View e reagisce eseguendo delle operazioni che, in alcuni casi, interessano la Model e che generalmente portano ad un cambiamento dello stato della View.

Git e Repository

Git è un software di controllo di versione. Si tratta di un'applicazione che permette, anche tramite una collaborazione decentralizzata, di tenere traccia di tutte le modifiche apportate al codice sorgente di un determinato programma (che si intende monitorare durante le fasi di sviluppo). Non è necessario un server centrale e gli sviluppatori possono collaborare tra loro in maniera parallela, senza essere per forza connessi in ogni istante. Git non è il solo sistema di controllo di versione esistente ma di certo è lo standard oggi universalmente riconosciuto in campo informatico.

Il creatore di Git è Linus Torvalds, il fondatore di Linux. Torvalds ha iniziato a sviluppare Git proprio come strumento di controllo per le sue personali attività di development sul kernel del sistema operativo.

Questo software viene idealizzato con il concetto di *branch* (rami). Il ramo principale, quello di partenza, viene detto *master*. Una registrazione di un nuovo stato del progetto viene detta ed effettuata con un *commit***.**

Comando status

Uno dei comandi più utilizzati è *status*. Esso consente di visualizzare lo stato corrente della Working Area e vedere, per esempio, quali file sono nella *Staging Area* pronti per essere inseriti nel prossimo commit.

Comando add

Per poter aggiungere un file o una modifica al *commit* bisogna utilizzare il comando *add*.

Tramite questo codice si aggiungono solamente i file ma non si è effettuato il commit: siamo ancora in una fase transitoria. Questo comando può essere usato in vari modi:

- Con i comandi "git add –A", "git add ." , "git add *" è possibile aggiungere tutti i file.

- Con il comando "git add *.txt" è possibile inserire tutti i file di un certo formato
- Con il comando "git add nome_file.txt" è possibile aggiungere uno specifico file

Comando commit

Per applicare le modifiche fatte si utilizza il comando *commit*, con i seguenti flag:

- "-a" per committare tutti i file in stato stage.
- "-m" per inserire il messaggio associato al commit che altrimenti verrebbe richiesto in una finestra interattiva.

Comando fetch

Per scaricare i commit e i relativi file si usa il comando *fetch*. Questo codice permette di scaricare le modifiche presenti in remoto senza nessun obbligo di unirle con quelle in locale.

Comando merge

Per unire le modifiche in locale con quelle in remoto viene utilizzato il comando *merge*. Questo valore permette di unire in maniera automatica le modifiche e di aggiornare il ramo.

In questa fase bisogna fare attenzione ai conflitti: questo può avvenire nel caso in cui due programmatori modifichino le stesse identiche righe di codice. Nel caso di questa eventualità bisognerà andare ad agire direttamente nel file specifico, risolvere i conflitti ed effettuare nuovamente il commit.

Comando push

Per inviare in remoto il commit appena effettuato con i relativi aggiornamenti, viene utilizzato il comando *git push*. Con questa istruzione, volendo, si può specificare anche il ramo su cui si vuole agire.

Comando pull

Per recuperare e scaricare il contenuto da un *repository* remoto e aggiornare immediatamente quello locale si utilizza il comando *pull*. Questo viene principalmente utilizzato quando in locale non sono presenti modifiche e si vuole aggiornare il proprio *repository* con i file in remoto.

BitBucket

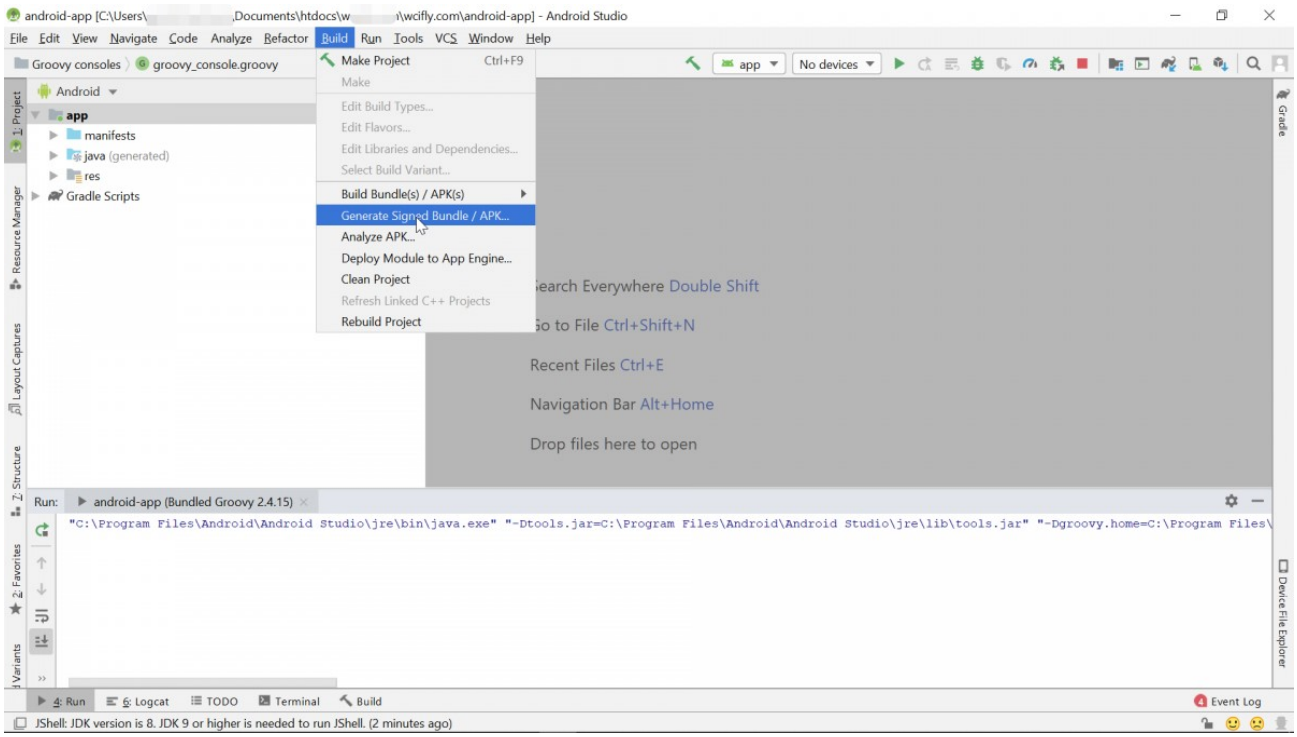
Come repository per il progetto si è utilizzato il servizio di hosting web-based, *Bitbucket*. Questo strumento oltre alla gestione del codice Git mette a disposizione altre funzionalità:

- Spazio in cui pianificare i progetti
- Collaborare su codici
- Effettuare test
- Implementare i risultati

Android Studio

Android studio è un ambiente di sviluppo integrato adibito alla creazione di applicazioni Android (*IDE, Integrated Development Environment*). È stato annunciato il 16 maggio 2013 in occasione della conferenza *Google*. Android Studio è disponibile gratuitamente sotto licenza *Apache 2.0.* Il download, basato sul software di *JetBrains IntelliJ IDEA*, è disponibile su *Windows*, *Mac OS X* e *Linux*. Sostituisce gli Android Development Tools (*ADT*) di *Eclipse*, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android.

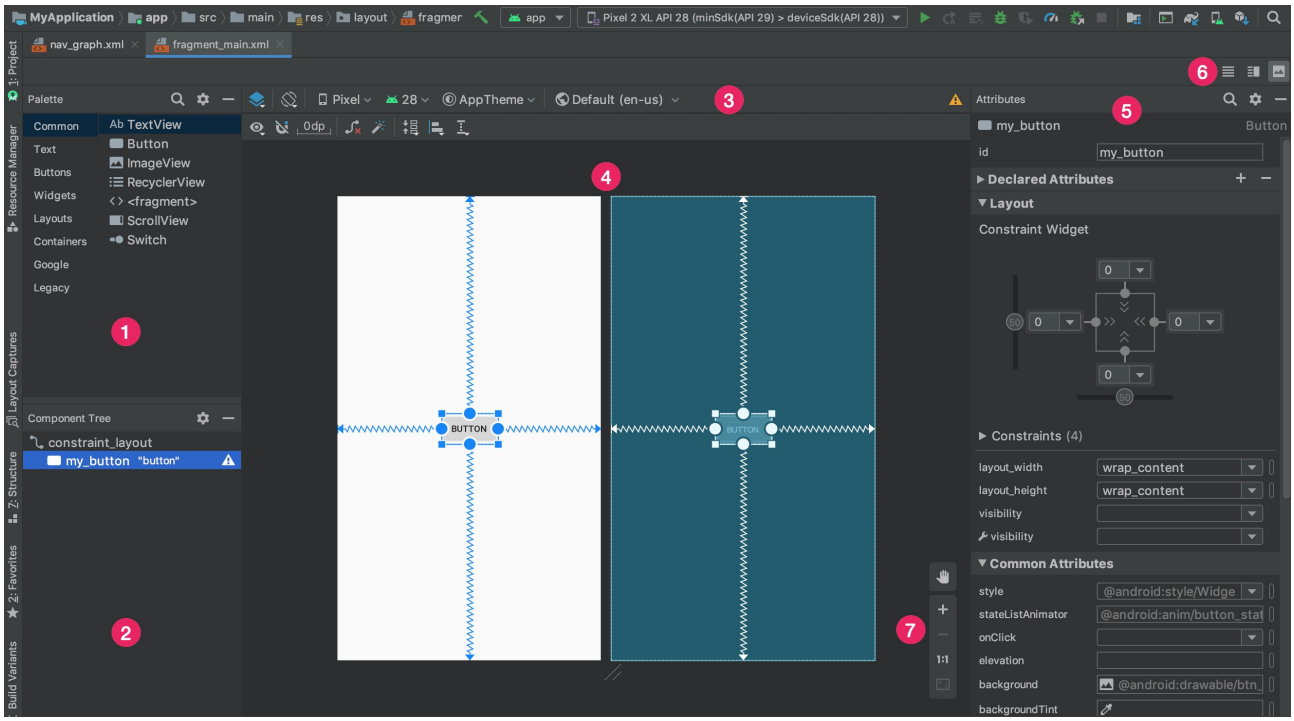
Android Studio è un IDE per il mondo Java particolarmente sensibile alle necessità dello sviluppatore. Al suo interno, include *Gradle*, un ottimo strumento per la build automation, molto flessibile ed erede di tutte le principali caratteristiche di predecessori come *Apache Ant* e *Maven*.
Per gli sviluppatori di mobile, Android Studio è uno strumento indispensabile per la compilazione, il debug del codice, la creazione di interfacce utente, la lettura della documentazione, la struttura del progetto e molto altro.



Sviluppare app Android, grazie a questo IDE, risulta semplice. L'ambiente di sviluppo proposto agli utenti è agevole ed è plasmato sulle esigenze del programmatore mobile. L'IDE proprietà di Google permette di testare l'applicazione, in fase di sviluppo, tramite un dispositivo fisico o, in alternativa, tramite un emulatore Android che simula i device virtuali (*Virtual Device, VD*). L'emulatore permette di testare l'applicazione su una vasta gamma di apparecchi e livelli di API Android, senza dover spendere cifre significative a livello economico per reperire i dispositivi fisici. Un *VD* fornisce varie funzionalità utili: simulare telefonate e messaggi in arrivo, specificare la posizione del dispositivo, riprodurre diverse velocità di rete e gestire tutti i sensori hardware. Tra le molteplici funzionalità previste dal software, una delle più utili è il *Debug*. Infatti è possibile eseguire l'applicazione nel device, fisico o virtuale, in *debugging.* Ciò permette di osservare l'esecuzione del codice in tempo reale, interrompere l'esecuzione del codice utilizzando i punti di interruzione, leggere i contenuti delle variabili e molto altro. Un'altra funzionalità indispensabile per l'uso quotidiano e di massa dell'IDE è quella dell'Assistant Editor che consiste nel completamento automatico durante la scrittura del codice o la generazione automatica di parte di esso.

Editor Layout

Android permette la creazione di interfacce grafiche sia procedurali sia dichiarative: possiamo creare un'interfaccia utente completamente in codice Kotlin/Java (metodo procedurale) oppure possiamo creare l'interfaccia utente attraverso un descrittore XML (metodo dichiarativo). Il metodo più utilizzato, però, è quello *ibrido*, in cui si crea un'interfaccia in modo dichiarativo e la si controlla e specifica in modo procedurale (si richiama il descrittore XML da codice e si continua a lavorare tramite programmazione). Un layout progettato in XML ricorda molto il linguaggio markup HTML per le pagine web. Ciò è particolarmente apprezzato da tutti quei programmatori che provengono da esperienze professionali o percorsi didattici nel settore. Per velocizzare la creazione di interfacce grafiche, Android Studio, mette a disposizione una funzionalità molto utile: *Editor Layout*. L'editor di layout consente di realizzare rapidamente grafiche trascinando gli elementi dell'interfaccia in un editor di progettazione visiva, senza dover scrivere manualmente codice XML. Permette una visualizzazione in anteprima del layout su diversi dispositivi e un facile ridimensionamento dell'interfaccia grafica per il funzionamento corretto su schermi di diverse grandezze.



1. *Tavolozza*: contiene vari elementi grafici o gruppi di essi, i quali possono essere trascinati o mossi nell'anteprima.
2. *Struttura dei componenti*: mostra la gerarchia dei componenti nel layout.
3. *Toolbar*: questi pulsanti permettono la configurazione dell'aspetto del layout e la modifica dei suoi attributi.
4. *Editor di progettazione*: permette di modificare la grafica in modo visivo e tramite trascinamenti.
5. *Attributi*: rappresenta un pannello di controllo dove è possibile modificare gli attributi corrispondenti all'elemento grafico selezionato.
6. *Modalità di visualizzazione*: è possibile scegliere la modalità di visualizzazione del layout. Le diverse opzioni sono: solo codice, solo progettazione visiva o entrambe.
7. *Controlli di zoom e panoramica*: possibilità di zoomare e spostare il layout all'interno dell'editor.

Presentazione Applicazione ADI Museum

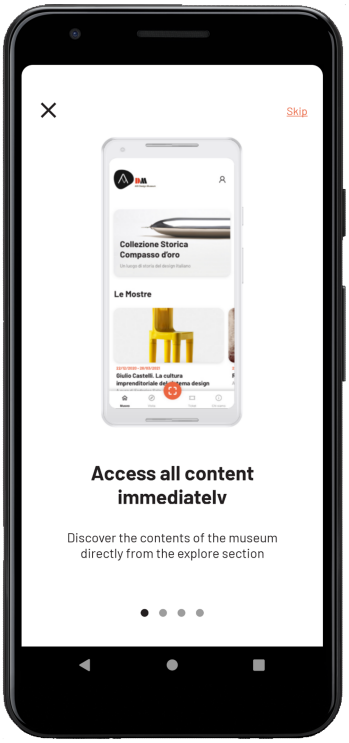
Molte delle conoscenze trattate in questo documento di tesi sono applicate nel progetto che andremo a illustrare, il quale prevede di sviluppare un'applicazione mobile nativa Android in linguaggio Kotlin. Nello specifico l'app è stata progettata su richiesta di ADI Museum Compasso d'Oro, il quale raccoglie a Milano gli oltre 350 pezzi premiati con il Compasso d'Oro. L'applicazione è fruibile da qualsiasi dispositivo mobile. Essa può essere utilizzata per accedere a delle funzionalità esclusive come la visualizzazione dei vari oggetti in esposizione, le mostre temporanee, una mappa interattiva delle aree del museo, l'accesso alle audioguide e l'acquisto online dei ticket (implementazione futura). Inoltre, per venire incontro ai visitatori internazionali, l'app supporta il multilingua (per ora italiano e inglese). Nei paragrafi successivi verrà mostrato il funzionamento dell'applicazione tramite foto ricavate dall'emulatore.

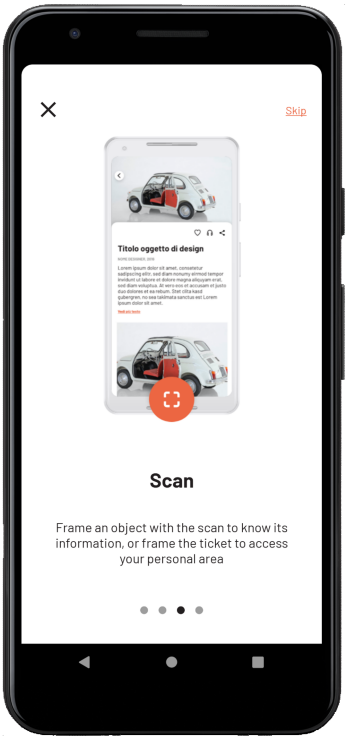
SplashScreen

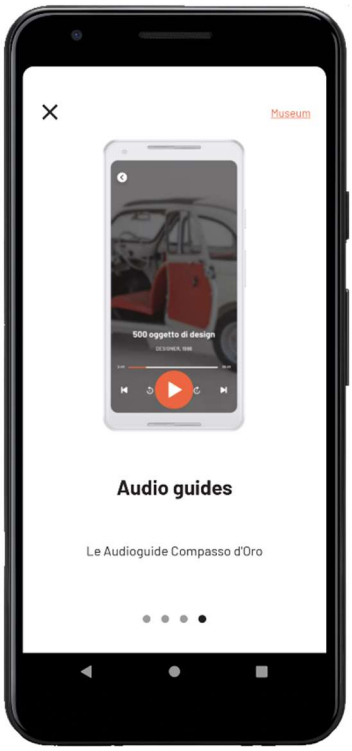
All'avvio dell'applicazione l'utente viene indirizzato alla pagina di SplashScreen. Questo tipo di Activity viene utilizzato per intrattenere l'utente durante il tempo dovuto al caricamento dei dati. E' molto importante che abbia una grafica accattivante per rendere gradevole l'attesa.

Helper

Alla prima apertura dopo l'installazione si aprirà un helper per guidare l'utente sulle varie funzionalità offerte dall'applicazione. Scorrendo tra le pagine si troverà un'introduzione alle principali funzioni fruibili tramite il software mobile.

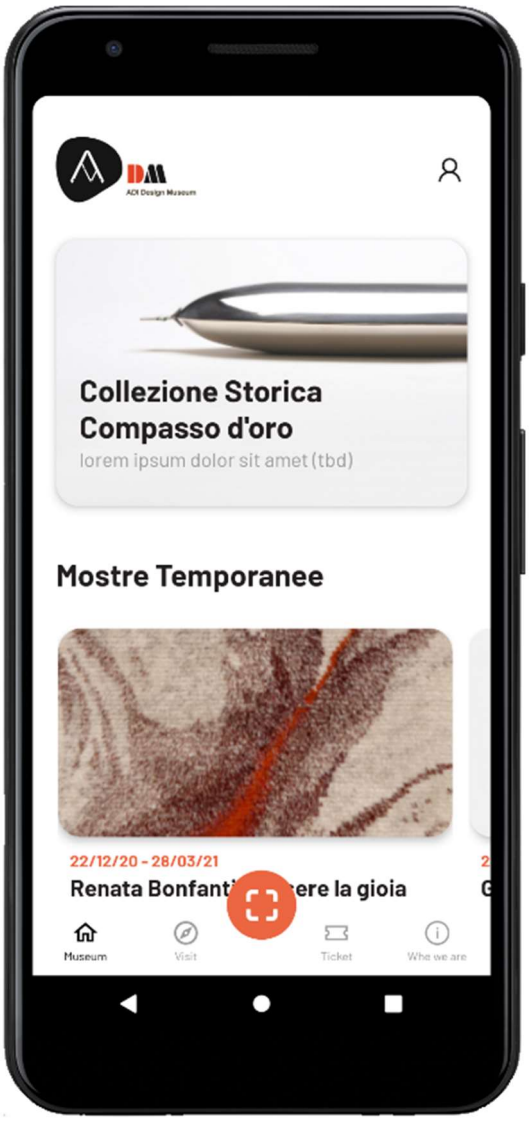


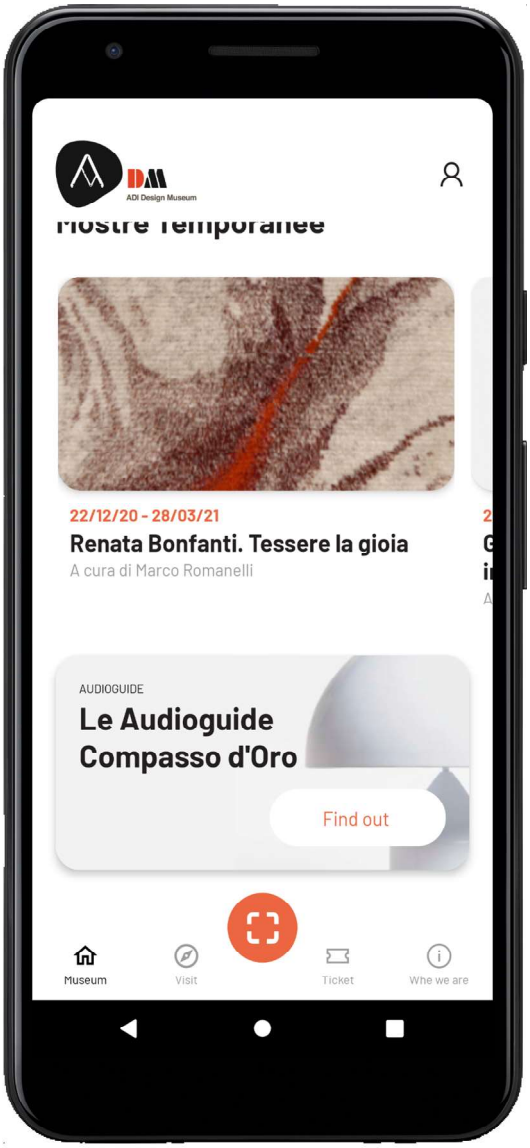




Pagina iniziale – "Museo"

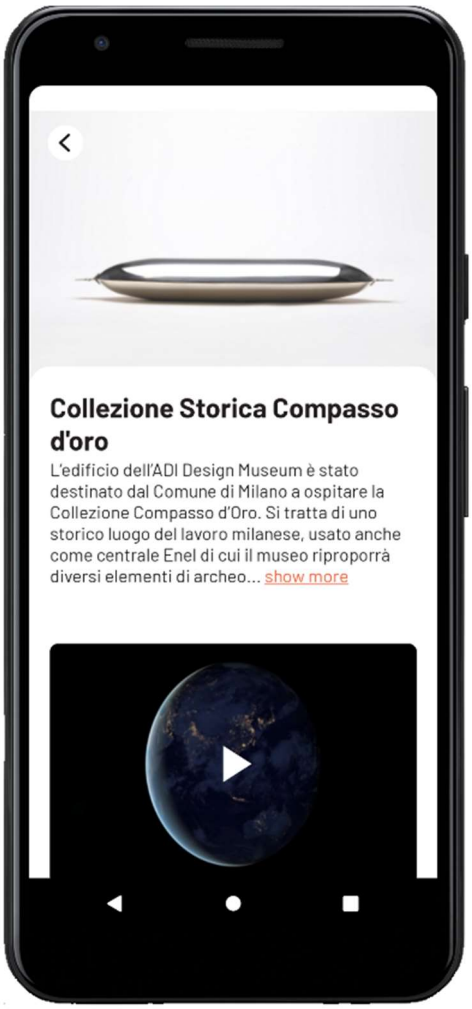
Alla conclusione dello SplashScreen si viene indirizzati alla pagina iniziale del museo nella quale è possibile avere accesso alla Collezione Storica, alle mostre temporanee (tramite uno *carousel* orizzontale) ed alla lista delle audioguide.





Collezione storica

Questa pagina contiene una presentazione del museo e alcuni cenni storici. Si è deciso di inserire un video ed alcune immagini di presentazione relativi ai contenuti del museo in modo da poter fornire al potenziale visitatore una panoramica completa della collezione.

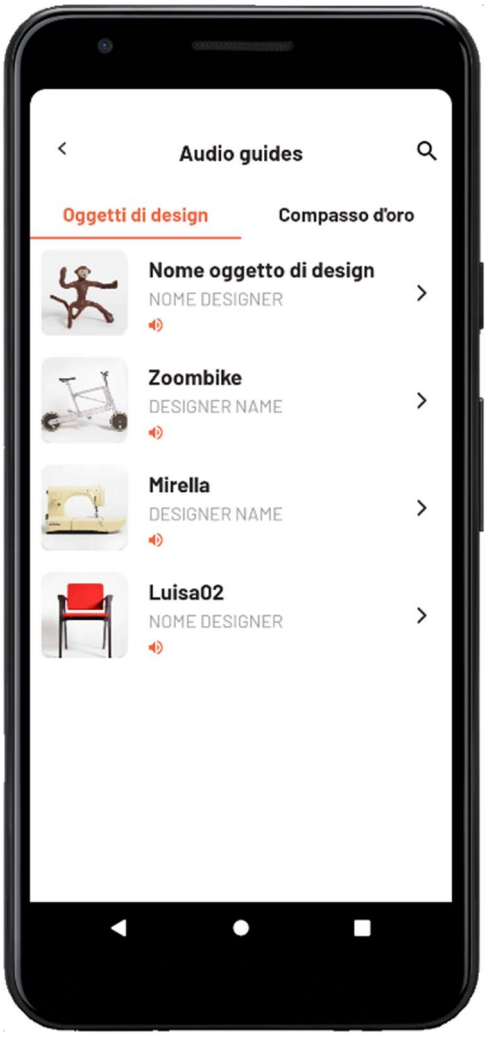


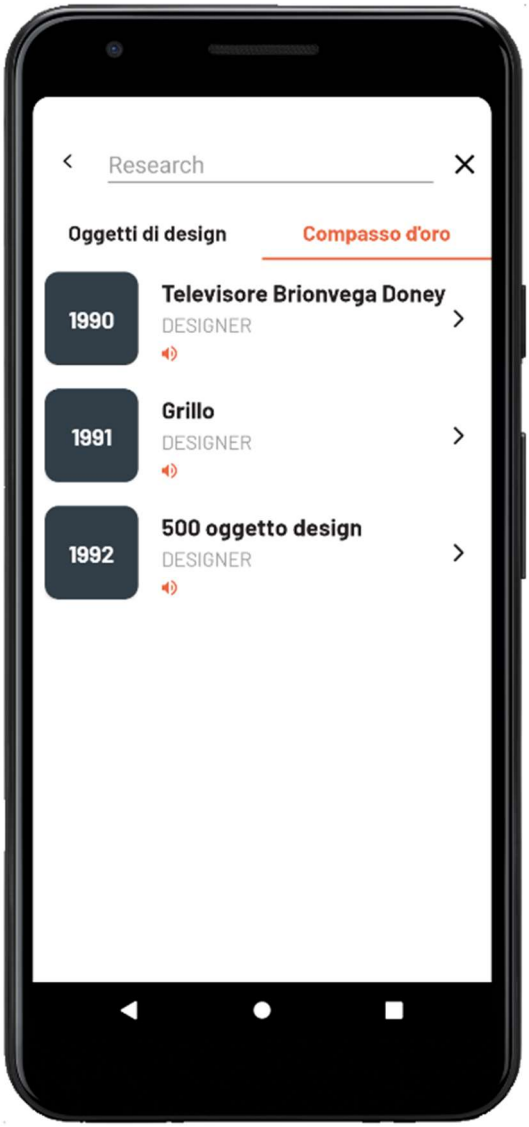
Mostra temporanea

In questa schermata è possibile visualizzare i dettagli di una specifica mostra temporanea. L'utente può reperire informazioni come durata e nome oltre ad una descrizione esauriente e immagini di *preview*.

Lista audioguide

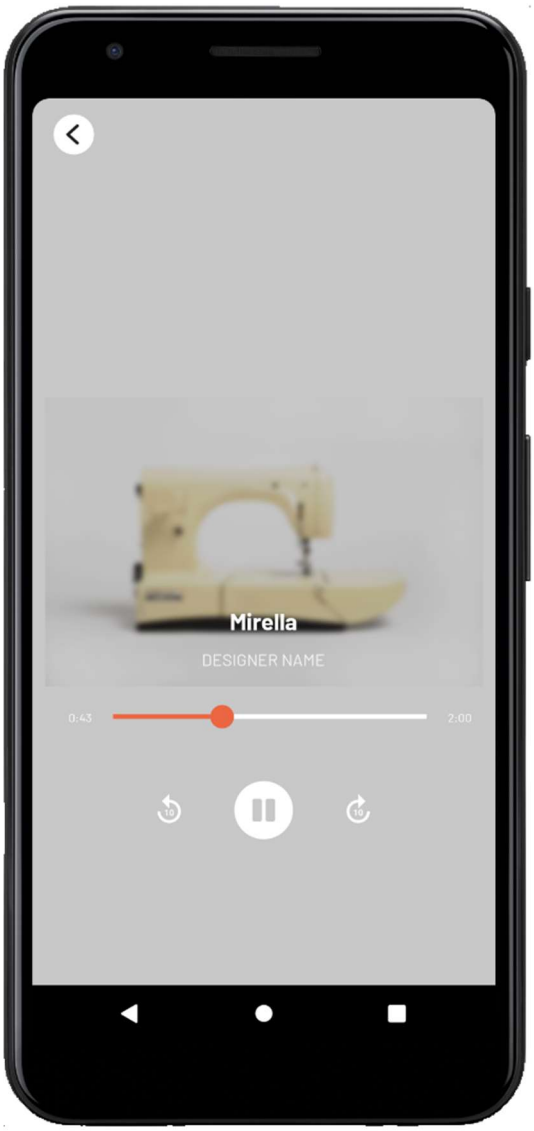
In questa pagina è possibile visualizzare la lista delle audioguide degli oggetti di design e dei Compassi d'oro (ovvero guide specializzate e divise per annate). È possibile ricercare l'audioguida più di interesse tramite la barra di ricerca, richiamabile dall'icona in alto a destra.

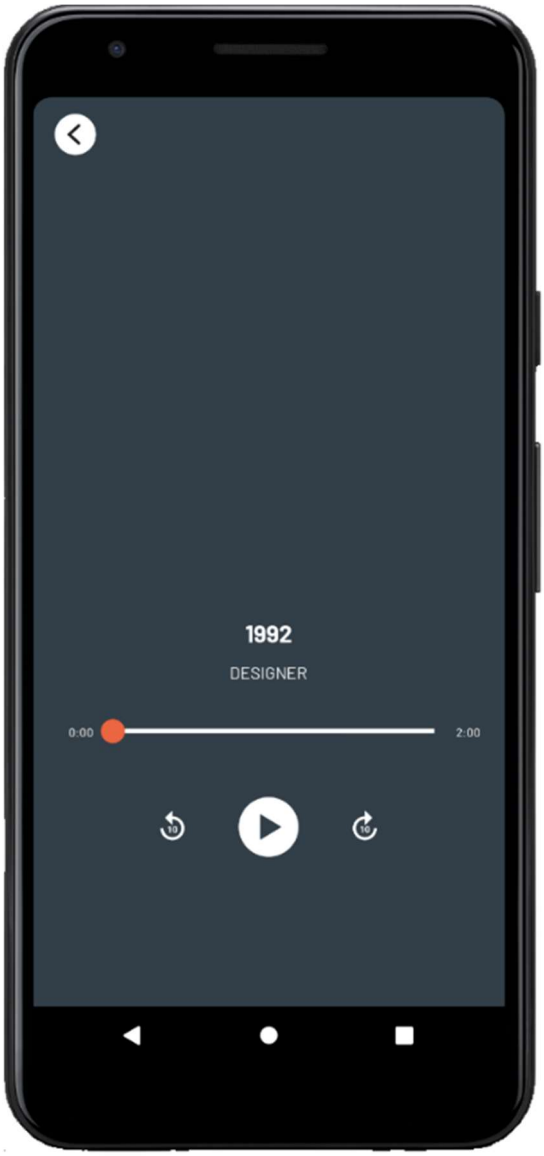




Audioguida

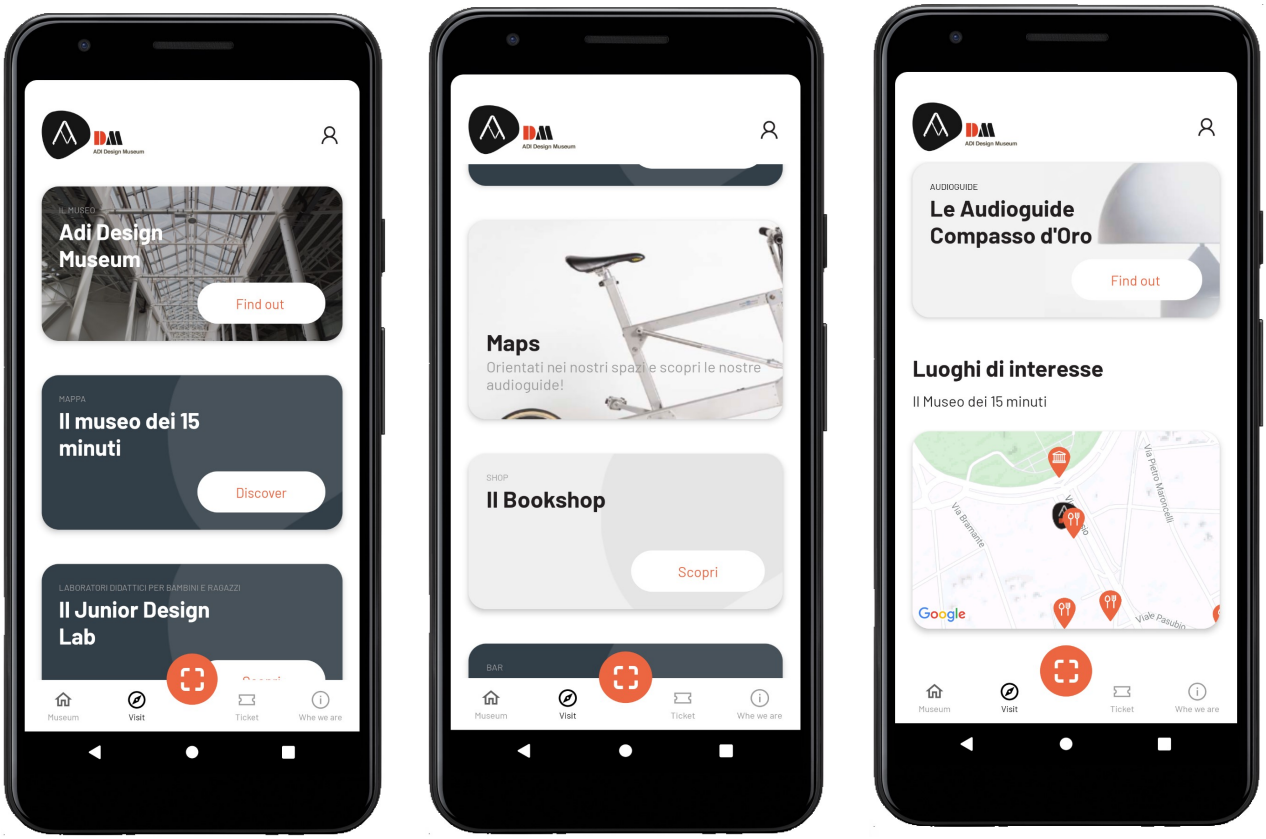
Questa activity rende possibile l'ascolto dell'audioguida selezionata con la possibilità di mettere in play e in pausa e di andare avanti e indietro di 10 secondi. La fruizione dell'audioguida viene facilitata dalla presenza di uno slider, con il quale è possibile visualizzare la durata e il tempo rimanente.





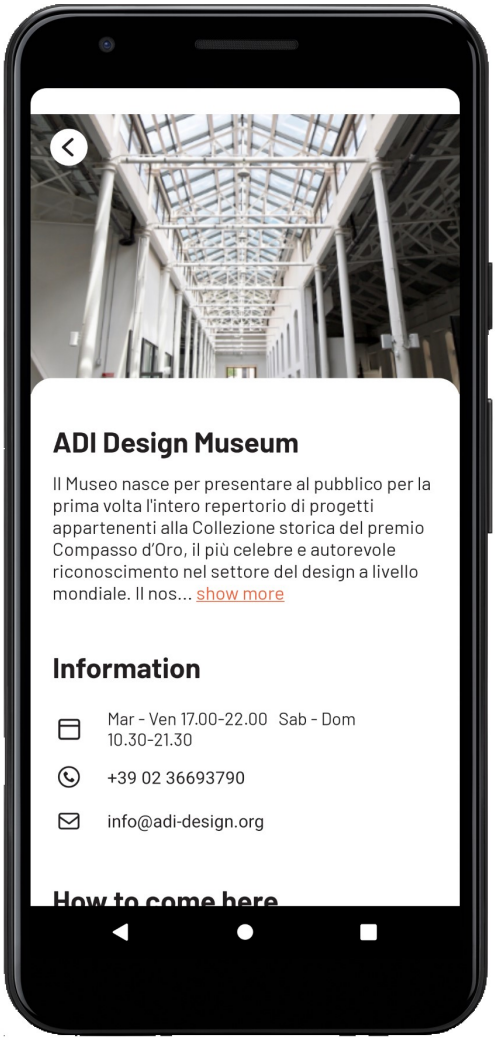
“Visita” – Voce di menù

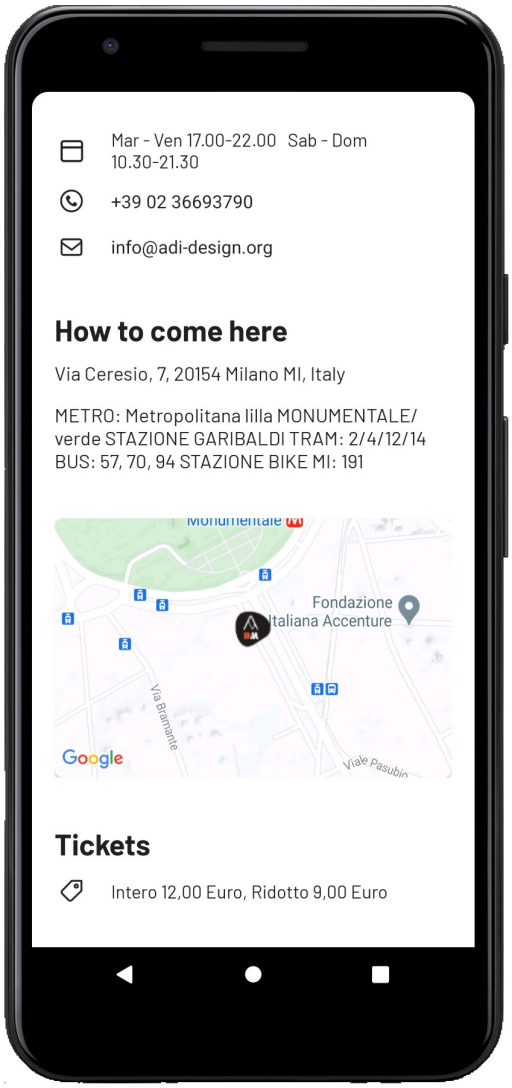
Facendo click su “visita” nel menù, sarà possibile visualizzare le sezioni utili per la visita del museo, come informazioni sulla localizzazione ed orari del museo. Inoltre fornisce info sui servizi accessibili all’interno dello stabile: caffetteria, bookshop e laboratori didattici per bambini e ragazzi. Durante la visita, tramite questa pagine, l’utente può accedere ad una mappa interattiva che indica la posizione delle opere esposte.

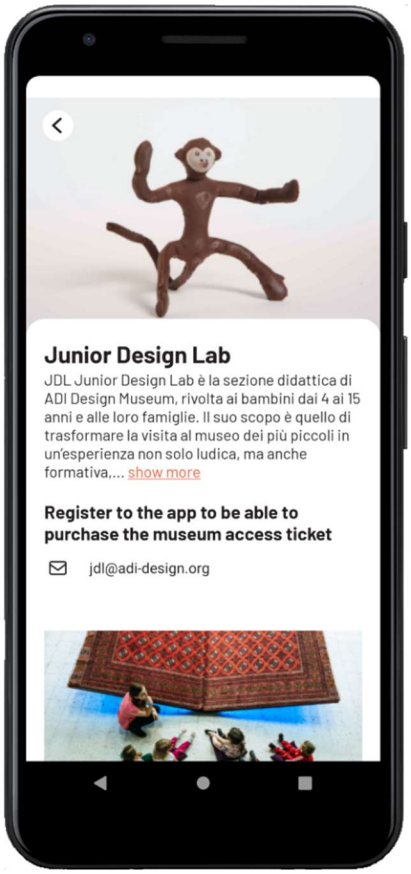


Adi Design Museum

In questa sezione sono presenti una serie di informazioni per facilitare l'arrivo al museo. Qui troviamo orari di apertura, recapito telefonico, indirizzo email, una mappa e il costo dei biglietti. È inclusa, inoltre, una breve introduzione alla storia del museo.







Junior Design Lab

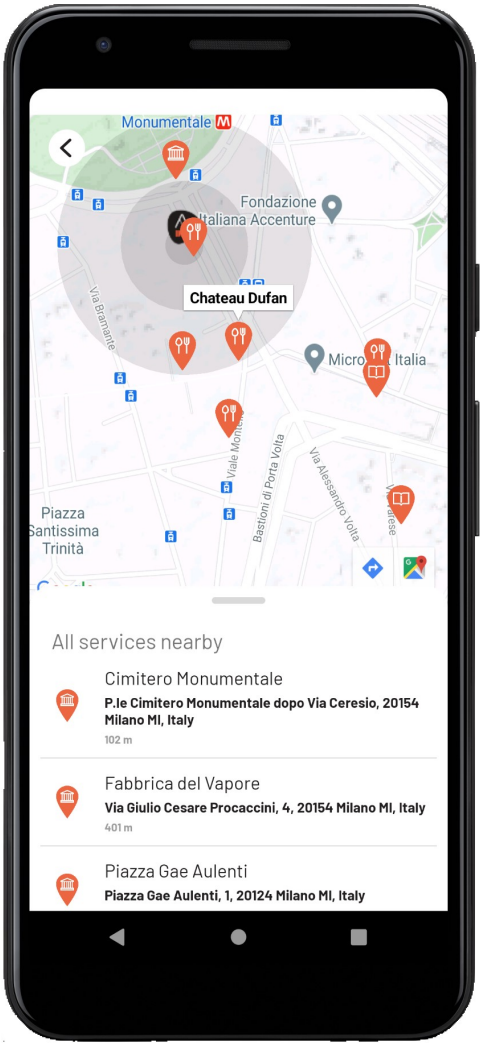
Questa pagina è dedicata ai laboratori didattici organizzati dal museo che coinvolgono bambini e ragazzi di età compresa tra 4 e 15 anni. È presente un indirizzo email per richiedere l'iscrizione, al click su di esso si aprirà, in maniera automatica, un servizio di posta elettronica per facilitare l'invio dell'email.

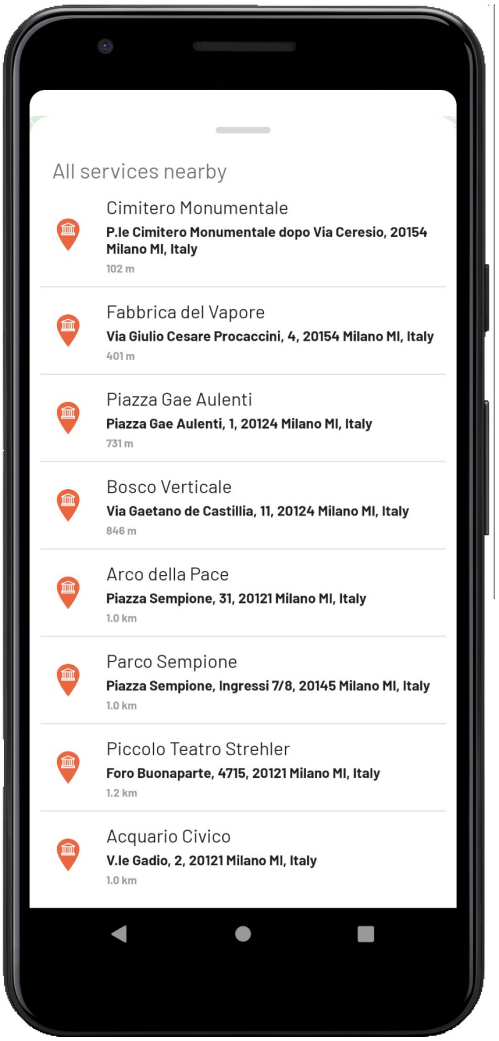
La mappa

Questa activity serve per facilitare l'orientamento all'interno del museo e per fornire una visita guidata a portata di smartphone. L'utente ha la possibilità di visualizzare la planimetria del museo con le opere maggiori collocate nelle posizioni corrispondenti. Al primo click su un oggetto la sua immagine si ingrandirà, al secondo, invece, l'utente verrà indirizzato al dettaglio dell'opera.

Luoghi di interesse

Tramite questa funzione l'utente può consultare una mappa interattiva dei dintorni del museo, nella quale sono indicati i servizi e le attività nelle vicinanze. Al tocco sul marker questo verrà selezionato e comparirà il nome del luogo corrispondente. Al di sotto del riquadro della mappa sarà possibile accedere ad una lista verticale dei posti segnati sulla cartina. Essa è completa di indirizzo, distanza e collegamento con Google Maps al click.



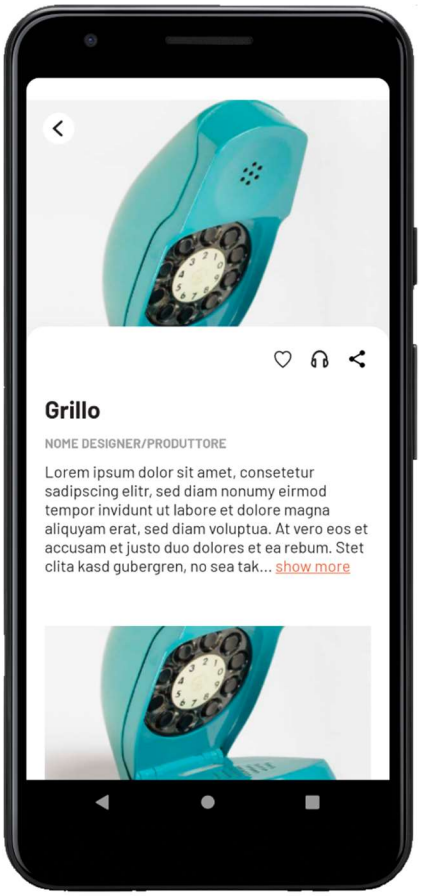


La caffetteria – Il Bookshop

Tramite l'utilizzo dell'app è possibile reperire informazioni relative ai servizi di caffetteria e bookshop presenti all'interno del museo. Nel caso della caffetteria, nella pagina è presente una breve descrizione corredata di immagini. L'activity del bookshop contiene, oltre a ciò, un collegamento allo shop online.

Dettaglio opera

In questa schermata vengono raccolte le informazioni relative all'opera selezionata, accompagnate da immagini di anteprima. L'utente può comodamente accedere alla pagina di ascolto dell'audioguida tramite l'apposita icona, a fianco può trovare inoltre un tasto per condividere i dettagli dell'opera. Nel caso in cui si sia effettuato l'accesso, sarà possibile aggiungere l'oggetto ai preferiti o rimuoverlo.

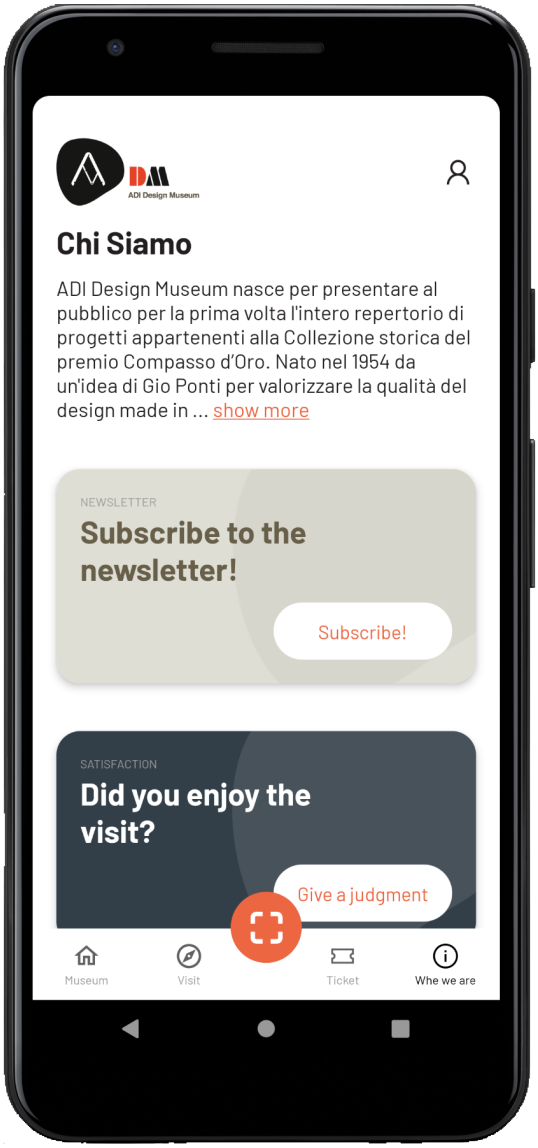


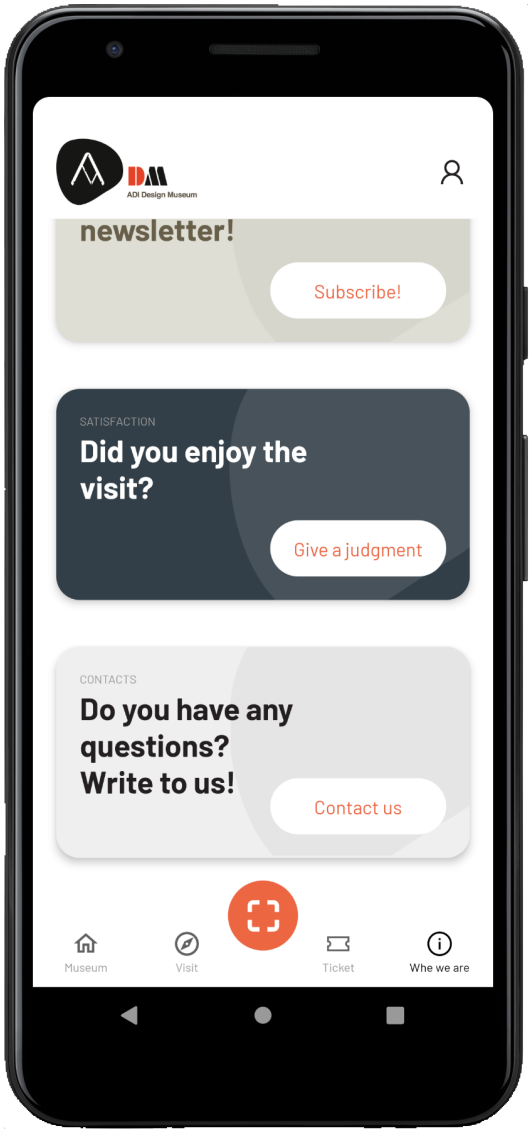
Scanner QrCode

Per rendere più immersiva la visita al museo, si è pensato di aggiungere la funzionalità di scannerizzazione di codici Qr. Di fianco a ciascuna opera esposta, sarà affisso un *QrCode* da poter inquadrare con l'applicazione in modo da accedere in tempo reale alla scheda relativa all'oggetto.

"Chi siamo" – Voce di menù

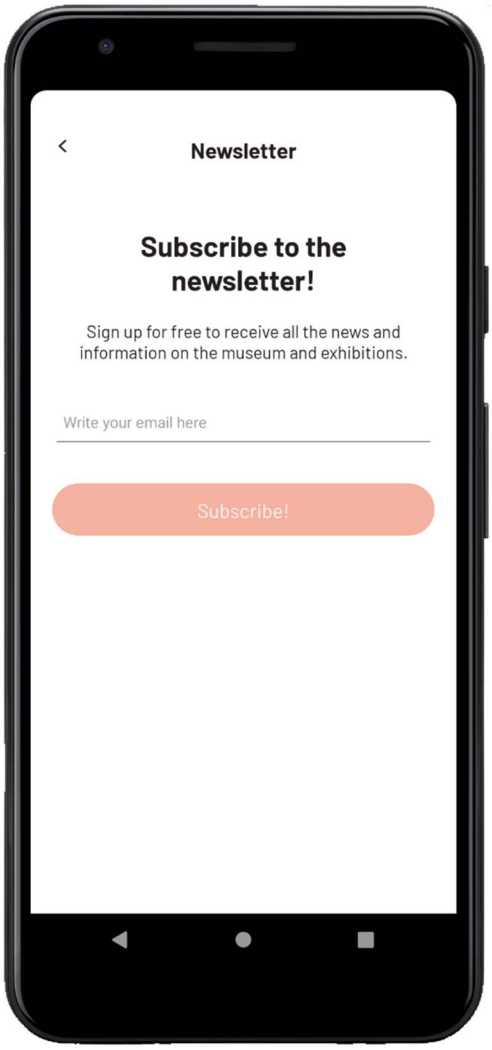
In questa sezione introduttiva si è scelto di inserire un breve compendio storico sulla nascita del museo. Offre inoltre la possibilità tramite tre piccoli layout di iscriversi alla newsletter, lasciare un feedback sulla visita ed infine di poter contattare via email la direzione del museo. Al click su quest'ultimo corrisponde un reindirizzamento alla propria app di servizio di posta elettronica.

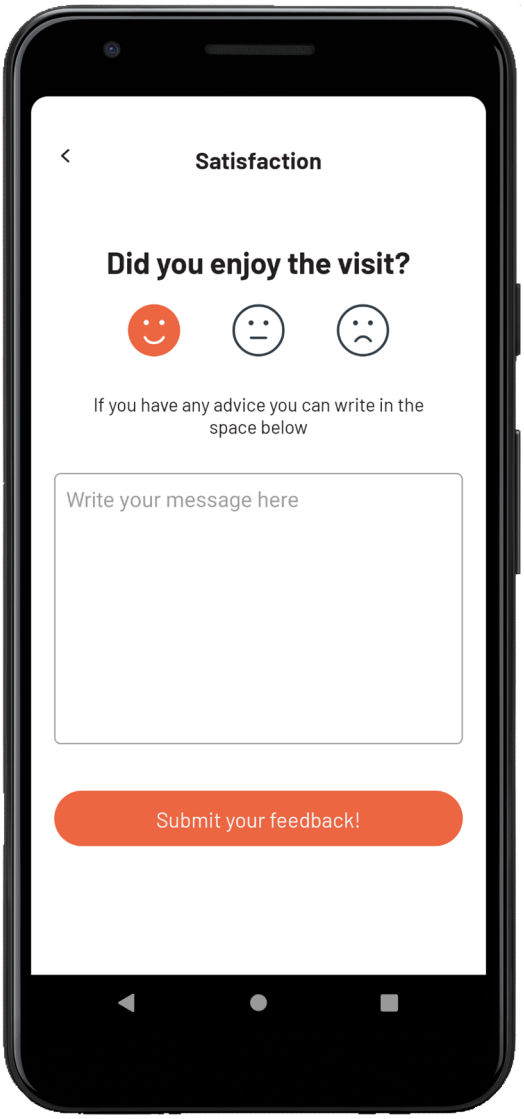




Newsletter

Tramite questa pagina è possibile iscriversi alla newsletter per rimanere sempre aggiornati sulle mostre in corso e le attività del museo. Per fare ciò è richiesto l'inserimento del proprio indirizzo email.



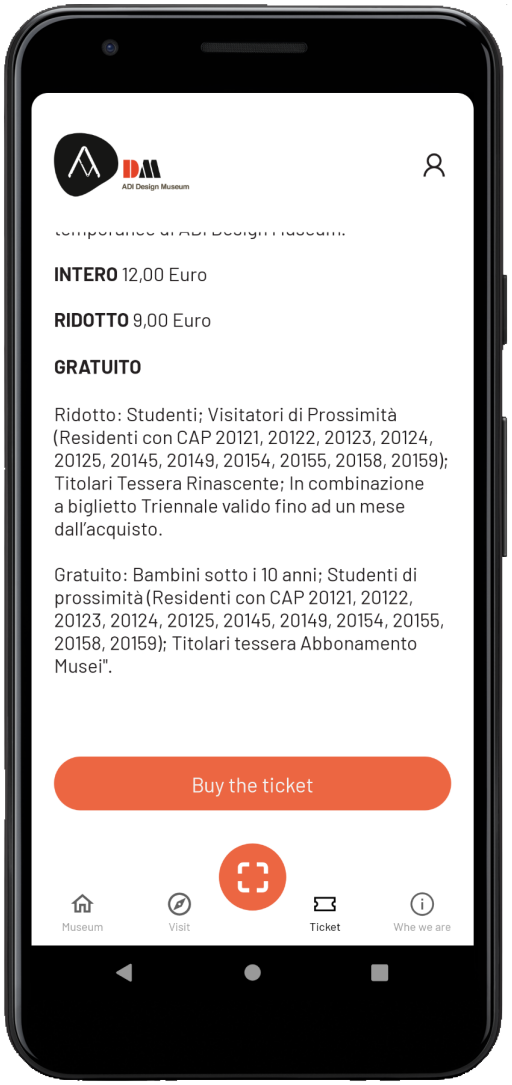


Gradimento

Questa activity permette di lasciare un riscontro e proporre suggerimenti utili per il miglioramento dell'esperienza. L'app propone un form composto da una valutazione sotto forma di emoticon e da un box di testo in cui l'utente è invitato a scrivere.

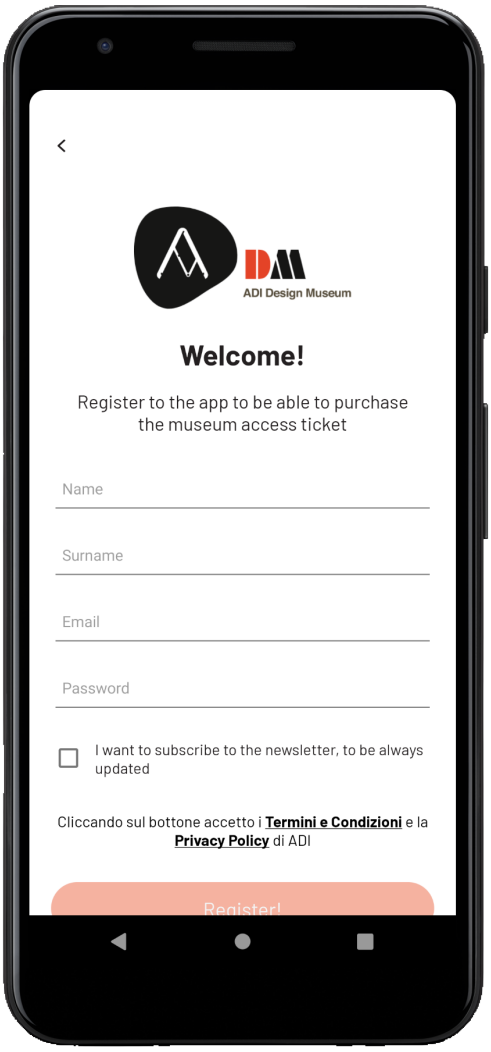
“Ticket” – Voce di menù

In questa pagina l’utente può visualizzare tutte le informazioni relative al costo del biglietto e le eventuali riduzioni a cui ha diritto. In fondo al testo è presente un pulsante che permette l’accesso alla schermata d’acquisto ai ticket.



Acquista biglietto

All'interno di questa activity, accessibile solo tramite previo login, è possibile acquistare direttamente i ticket online. La schermata offre la possibilità di selezionare il tipo di biglietto, la data, l'ora e conoscere anticipatamente il prezzo totale.

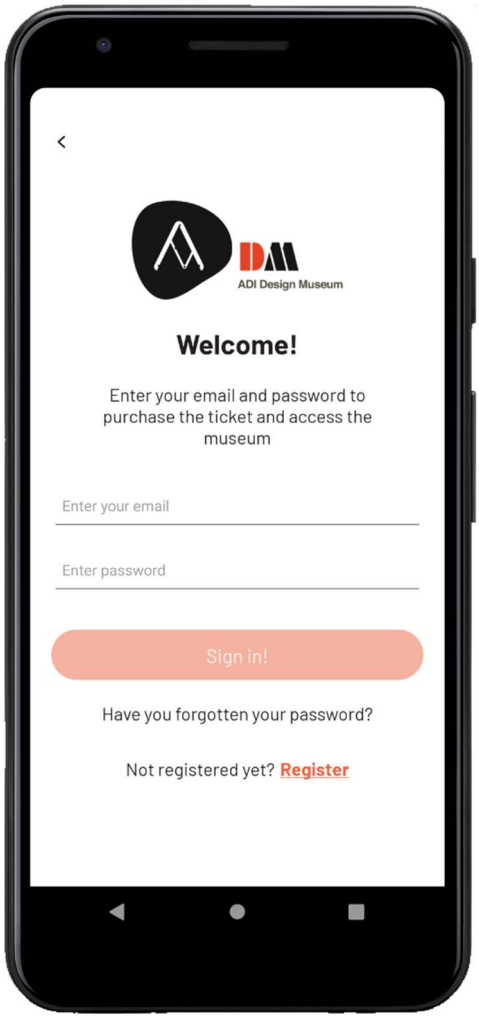


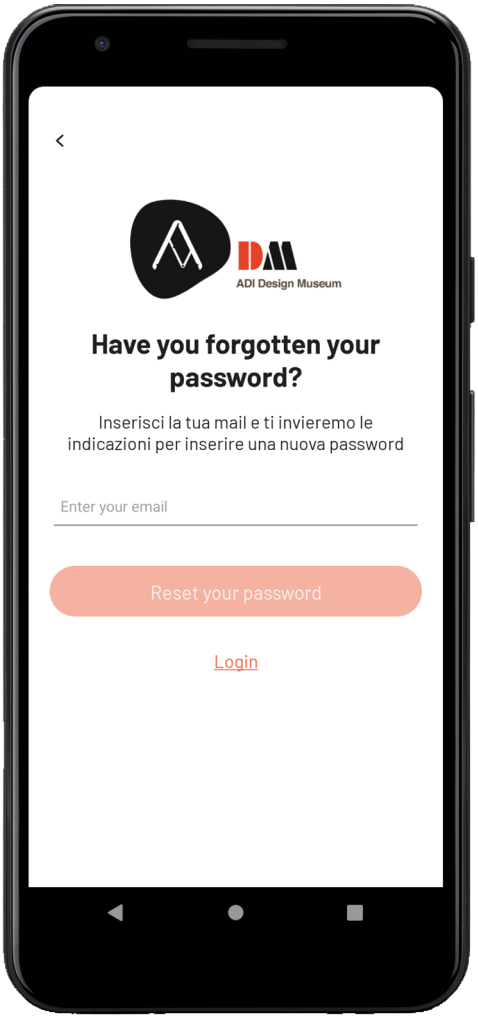
Registrati

Per poter accedere ai contenuti esclusivi dell'app e all'area privata del proprio profilo, è necessario iscriversi. L'iscrizione avviene tramite un form che richiede nome, cognome, indirizzo di posta elettronica e password. Inoltre, è possibile iscriversi alla newsletter spuntando la casella corrispondente.

Accedi

Se si è già in possesso di un account all'interno dell'applicazione, tramite questa pagina è possibile effettuare l'accesso inserendo email e password. Vi è inoltre l'opzione di recuperare la propria password tramite la schermata successiva.



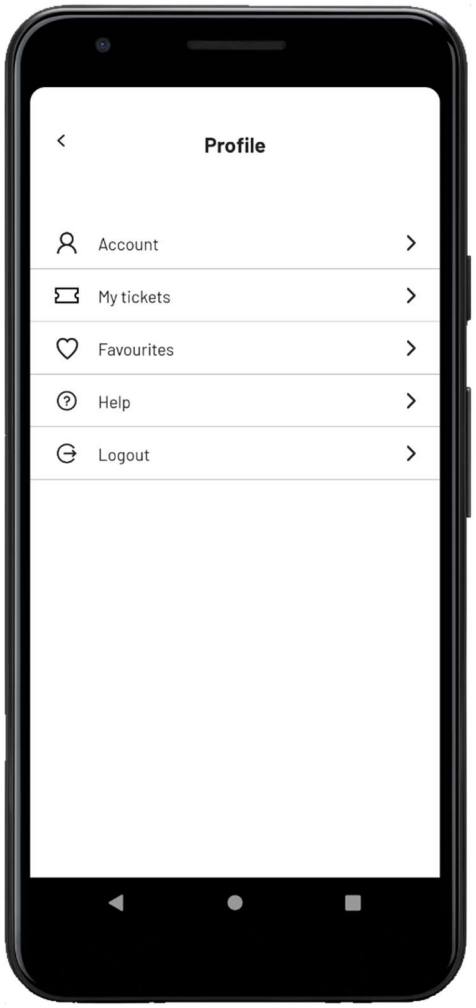


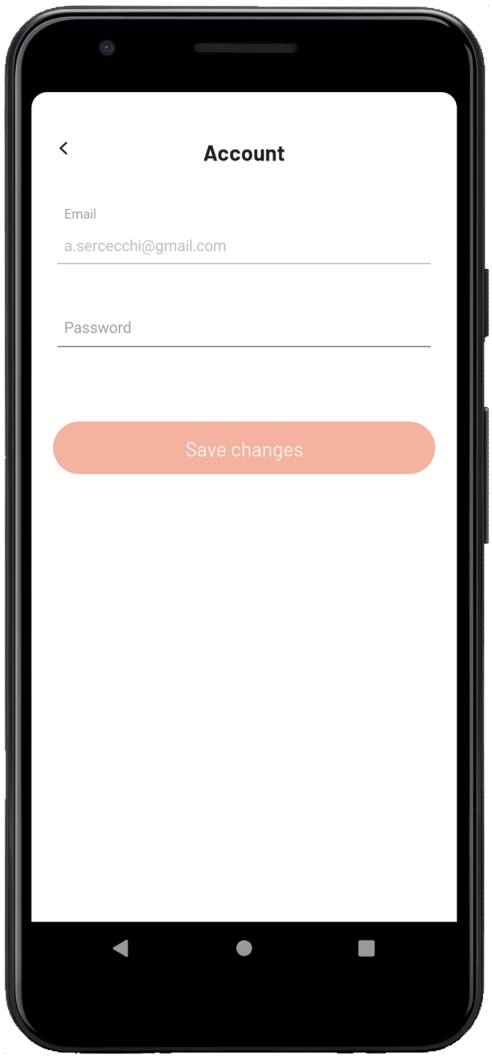
Recupero password

Nell'eventualità di una password dimenticata è possibile richiederne una nuova. Inserendo il proprio indirizzo di posta elettronica nell'input fornito, verrà inviata un email con il link per il recupero.

Profilo

Una volta effettuato l'accesso, sarà disponibile un'area privata in cui si avrà la possibilità di cambiare le proprie credenziale, visualizzare i ticket in possesso, accedere ai preferiti, richiamare l'helper e disconnettersi. Approfondiremo, in seguito, le prime tre funzioni.



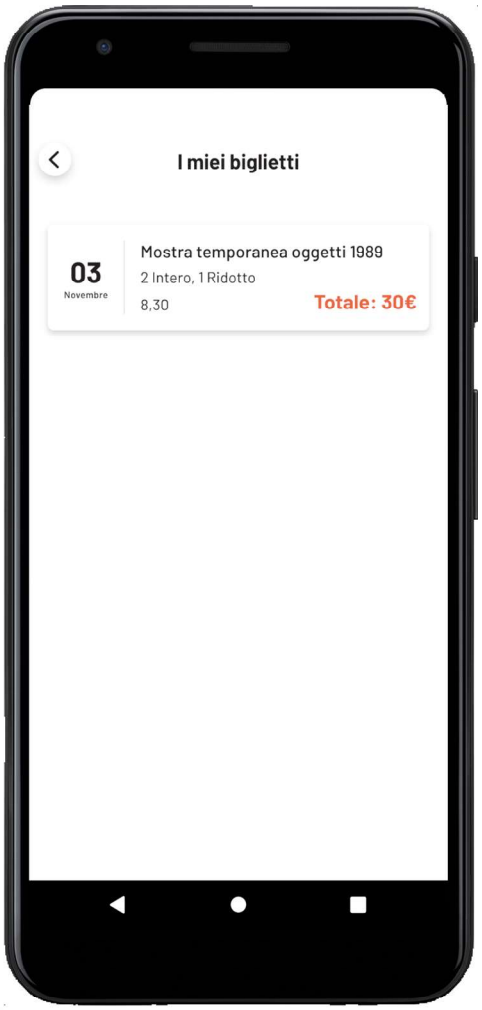


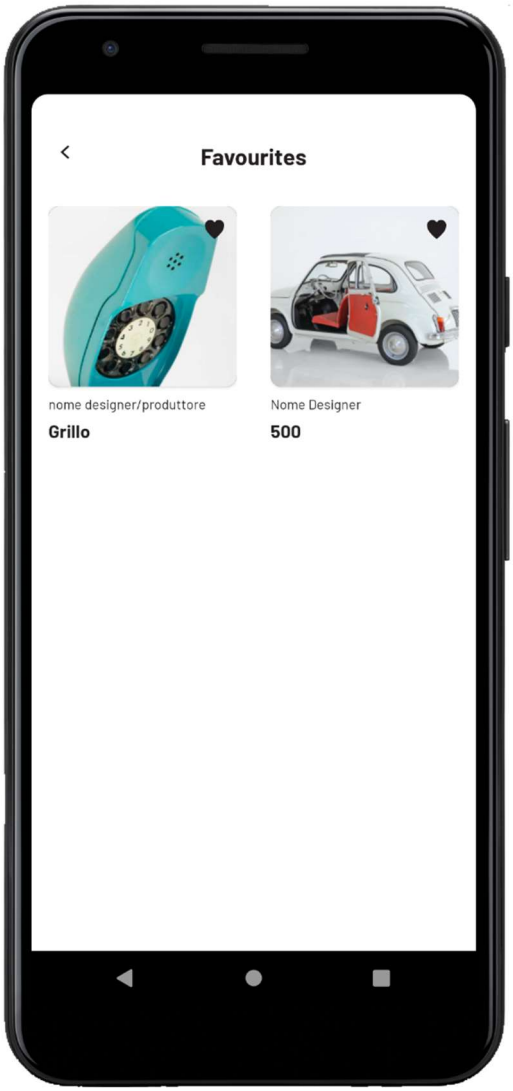
Account

Tramite questa pagina è possibile visualizzare l'email associata al profilo e aggiornare la propria password.

I miei biglietti

In questa activity è possibile visualizzare la lista dei ticket in proprio possesso. Essa è completa di data, orario, tipo e numero dei biglietti ed il prezzo totale pagato.





Preferiti

Accedendo a questa schermata si avrà la possibilità di visualizzare l'elenco degli oggetti preferiti. Al click su un elemento si verrà reindirizzati nella sua pagina dettaglio. Cliccando l'icona del cuoricino, invece, l'opera corrispondente verrà eliminata dai favoriti.

Sviluppi futuri

Per completezza si elencano qui di seguito le future implementazioni, ancora in fase di analisi ed elaborazione:

- Pagamento online tramite servizi esterni, già in fase di sviluppo per concludere l'applicazione prima del rilascio ufficiale.
- Possibilità di effettuare l'accesso tramite account Google e Facebook.
- Creazione automatica di un file pdf contente il biglietto cartaceo che verrà inviato per email al suo acquisto.
- Possibilità di visualizzare il menù della caffetteria tramite l'applicazione.
- Disponibilità, all'interno dell'app, del catalogo del Bookshop.
- Aggiunta di un calendario aggiornato in tempo reale dei vari laboratori per ragazzi.

Conclusioni

Lo svolgimento della presente tesi ha permesso di ampliare le mie conoscenze, in particolare quelle relative allo sviluppo di applicazioni mobile Android attraverso il linguaggio Kotlin. Questo è stato possibile tramite studi di approfondimento ed una continua ricerca della risoluzione ottimale delle problematiche. Grazie allo stage universitario effettuato presso l'azienda "Be Ready Software", sono riuscito a completare il lavoro in maniera autonoma e dinamica. Inoltre il mio correlatore è stato sempre presente in caso di necessità e per riscontri settimanali sull'andamento del lavoro. Questa esperienza mi ha permesso, a livello professionale, di prendere coscienza delle mie attitudini, qualità e mancanze da colmare.

In conclusione, mi ritengo soddisfatto del lavoro effettuato durante lo svolgimento della tesi di laurea, in quanto mi ha arricchito a livello formativo ma soprattutto a livello umano. A testimonianza del bel rapporto professionale sviluppatosi con il Dott. Benedetti Daniele, la Dott.ssa Benedetti Manuela e tutti i colleghi, mi è stato proposto di continuare il rapporto lavorativo.

Bibliografia

- Android Studio 4.1 Development Essentials – Neil Smyth
- Sviluppare App per Android - Harvey M. Deitel, Paul J. Deitel
- Android 9. Guida completa per lo sviluppo di applicazioni mobile – Massimo Carli
- Kotlin. Guida al nuovo linguaggio di Android e dello sviluppo mobile. – Massimo Carli

Sitografia

- Android Developers - https://developer.android.com/
- Html.it - https://www.html.it/
- Androidiani.com - https://www.androidiani.com/
- Stack Overflow - https://stackoverflow.com/
- Italian Coders - https://italiancoders.it/
- MRW - https://www.mrw.it/

Ringraziamenti

Vorrei ringraziare innanzitutto il relatore di questa tesi, il professor Fausto Marcantoni, per la disponibilità, gentilezza e attenzione dimostrata durante la stesura del lavoro, oltre che per la passione e coinvolgimento provati all'ascolto di ogni sua lezione.

Un sincero grazie va all'azienda, Be Ready Software, che mi ha permesso di lavorare e sviluppare questo interessante progetto. In particolare, un ringraziamento va al correlatore di questa tesi, Daniele Benedetti, che mi ha seguito costantemente nello sviluppo dell'applicazione.

Un grazie speciale va a tutte quelle persone che mi hanno accompagnato in questo percorso di studi: i professori, i miei compagni universitari, i miei amici ma soprattutto la mia famiglia e la mia fidanzata i quali mi hanno sempre sostenuto.

È stato un percorso difficile e intenso, per questo sono orgoglioso di averlo concluso, anche grazie a voi.