

# Approccio CLIENT-SERVER

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

Le richieste degli smatphone sono gestite da un server multithread strutturato logicamente in 3 parti (classi) :

- MasterServer
- MateriaPlusUpdater
- SlaveThreadServer

# Approccio CLIENT-SERVER

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

Le richieste degli smatphone sono gestite da un server multithread strutturato logicamente in 3 parti (classi) :

- MasterServer  
Thread principale che gestisce una socket in ascolto, ogni volta che riceve una richiesta di connessione, genera un Thread secondario che dovrà gestire la richiesta del client per poi rimettersi in ascolto
- SlaveThreadServer
- MateriaPlusUpdater

# Approccio CLIENT-SERVER

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

Le richieste degli smatphone sono gestite da un server multithread strutturato logicamente in 3 parti (classi) :

- MasterServer
- SlaveThreadServer

Riceve dal client la sua laurea ed in base a questa gli risponde inviandogli una materia appropriata, ovvero una classe 'MateriaPlus' in formto Json.

- MateriaPlusUpdater

# Approccio CLIENT-SERVER

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

Le richieste degli smatphone sono gestite da un server multithread strutturato logicamente in 3 parti (classi) :

- MasterServer
- SlaveThreadServer
- MateriaPlusUpdater

Esegue il seguente loop:

- Rispristina delle vecchie materie se il server è stato riavviato ma il loro tempo di vita non è scaduto
- Riceve da MainMateriaPlusCalculator le materie che gli SlaveThreadServer dovranno inviare ai client
- Si mette in attesa per un tempo pari al tempo di vita delle materie che sono state generate

# Algoritmo di calcolo della materia

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

La classe MainMateriaPlusCalculator si occupa, con l'ausilio della classe SlaveMateriaPlusCalculator, di generare su richiesta una materia con un voto per ogni corso di laurea con le seguenti specifiche:

- Le materie siano randomiche
- I voti vengano generati tutti con lo stesso attributo di rarità anche in caso di materie con crediti differenti
  - Possibile tramite la funzione:
$$Rarity = \frac{2}{3} * [(Voto - 17) / 14] + \frac{1}{3} * [Crediti / 18]$$
  - Livello di rarità scelto randomicamente fra 5 possibili
- Abbiamo tutti la stessa posizione
- La materia e/o la posizione dei nuovi esami generati devono essere differenti da quelli della richiesta precedente

# Algoritmo di calcolo della materia

Server

Tolleranza ai guasti

Highlight Codice

Client

Splash Screen

```
public MateriaPlus[] getMateriePlus() {
    int parameter=1;
    int rarity;
    int indexFor=0;
    Double[] latAndLng=new Double[2];

    //calcolo la rarità
    rarity=slave.foundRarity();

    //chiedo lat e lng
    while(parameter==1){
        latAndLng=slave.requestPos();
        if(latAndLng!=lastLatAndLng)
            parameter=0;
    }

    //preparo la materia
    for (MateriaPlus materia: materie){
        //setto la laurea
        materia= new MateriaPlus( subject: null, credits: 0, mark: 0, rarity: 0);

        materia.setLaurea(nomeLauree[indexFor]);

        //prendo il voto dal file
        materia=slave.requestMateria(materia);

        //setto la posizione
        materia.setLat(latAndLng[0]);
        materia.setLng(latAndLng[1]);

        //setto la rarità
        materia.setRarity(rarity);

        //calcolo il voto
        materia=slave.setVoto(materia);
        materie[indexFor]=materia;
        indexFor++;
    }
}
```

# Algoritmo di calcolo della materia

Server

Tolleranza ai guasti

Highlight Codice

Client

Splash Screen

```
//setto per evitare di riproporre stesse materie o posizione
for (int i=0;i<LAUREE;i++){
    ultimaMateria.put(materie[i].getLaurea(),materie[i].getSubject());
}
lastLatAndLng=latAndLng;

return materie;
}

/**Per un futuro diventerebbe più efficiente passare ad
 * un HashTable se gli indirizzi di laurea aumentano*/
public Hashtable hashMateriePlus(){
    Hashtable<String,MateriaPlus> materiaHash= new Hashtable<String, MateriaPlus>( 3){};
    MateriaPlus[] materione= getMateriePlus();
    materiaHash.put(LAUREA_ING_INFO,materione[0]);
    materiaHash.put(LAUREA_ECO,materione[1]);
    materiaHash.put(LAUREA_MED,materione[2]);
    return materiaHash;
}
```

# Riavvio del Server

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

Nel caso in cui il server dovesse esser soggetto a dei guasti e quindi sia necessario riavviarlo, le materie precedentemente generate possono essere recuperate. Difatti la classe 'MateriaPlusUpdater' ogni volta che chiede delle nuove materie, le salva (in formato Json) su un file, dal quale è possibile calcolare il tempo di vita rimanente delle materie in esso memorizzate. Se è un tempo strettamente positivo, le materie vengono ripristinate.



# MasterServer

Server

Tolleranza ai guasti

Highlight Codice

Client

Splash Screen

```
//quanto viene accettata una nuova connessione, resetta la socket di ascolto,  
//sara' compito del thread gestire la connessione con l'ultimo client  
//che ha richiesto la connessione  
Socket clientSocket = null;
```

```
try {  
    clientSocket = this.serverSocket.accept();  
} catch (IOException e) {  
    if(isStopped()) {  
        System.out.println("Server Stopped.");  
        return;  
    }  
    throw new RuntimeException(  
        "Error accepting client connection", e);  
}
```

```
//appena ho una richiesta di connessione, dopo averla accettata, verra' gestita  
//da un altro thread (SlaveThreadServer)  
SlaveThreadServer slaveRunnable = new SlaveThreadServer(clientSocket);
```

```
Thread salveThreadServer = new Thread(slaveRunnable);
```

```
salveThreadServer.start();
```

# SlaveThreadServer

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

```
inputSocket = clientSocket.getInputStream();
dataRead = new DataInputStream(inputSocket);

//leggo dal client il tipo di laurea della materia che sta richiedendo
laurea = dataRead.readUTF();

outputSocket = clientSocket.getOutputStream();
dataWrite = new DataOutputStream(outputSocket);

System.out.println("Client "+clientSocket.getInetAddress().getHostAddress()+" has sent:" +laurea);

//prendo la materia in base alla laurea
if(laurea.compareTo(LAUREA_ING_INFO) == 0){
    requestedMateriaPlus = lookForMateriaPlus(LAUREA_ING_INFO);
}else if(laurea.compareTo(LAUREA_ECO) == 0){
    requestedMateriaPlus = lookForMateriaPlus(LAUREA_ECO);
}else if(laurea.compareTo(LAUREA_MED) == 0){
    requestedMateriaPlus = lookForMateriaPlus(LAUREA_MED);
}else{
    //se sono qui la laurea inviata non corrisponde a nessuna di quelle
    //memorizzate, ramo che non verrà mai eseguito
    clientSocket.close();
    return;
}
```

# MateriaPlusUpdater, tolleranza ai guasti

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

```
buffRead = new BufferedReader(new FileReader(recoverFile));  
materiePlusJson = buffRead.readLine();
```

```
if(materiePlusJson != null) {
```

```
    //controllo che il tempo di emissione di una delle vecchie materie  
    //(che è salvata su un file) più il tempo di vita di default meno il tempo attuale  
    //sia zero (CONTROLLO EFFETTUATO SUI MINUTI, per essere un po' più laschi)  
    materie = gson.fromJson(materiePlusJson, MateriaPlus[].class);
```

```
    emissionPlusTTL = sumTimeInt(materie[0].getEmissionTime(), MATERIA_TIME_TO_LIVE_MINUTES);  
    actualTime = getCurrentTimeUsingCalendar();
```

```
    difference = timeDifference(emissionPlusTTL, actualTime);
```

```
    //Se tale differenza è minore di zero, il server è andato giù e posso ancora  
    //emettere le vecchie materia con il tempo rimanente
```

```
    //la differenza massima ammessa è del tempo di vita della materia,  
    //se è maggiore, significa che sono passato all'ora del giorno successivo,  
    //per esempio: emissionTime = 20:40:00, actualTime = 11:00:00, la loro differenza è maggiore di  
    //ma si è passati al giorno successivo
```

```
    if (difference < MATERIA_TIME_TO_LIVE_MINUTES*60*1000 && difference > 0) {
```

```
        //aggiorna le materie solo se nessun altro client le sta leggendo
```

```
        MasterServer.readWriteLock.writeLock().lock();
```

```
        MasterServer.materiePlus = materie;
```

```
        //unlock
```

```
        MasterServer.readWriteLock.writeLock().unlock();
```

```
        for (MateriaPlus materia : materie)
```

```
            System.out.println("Json Materia recovered:\n" + materia.toString());
```

```
        //fatto tutto, si mette in attesa per tanto tempo quanto rimane alle materie
```

```
        System.out.println("Difference: "+difference);
```

```
        Thread.sleep(Math.abs(difference));
```

```
    }
```

# MateriaPlusUpdater, richiesta materie

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

```
while(true) {
    materie = MainMateriaPlusCalculator.getInstance().getMateriePlus();

    //setto il tempo di emissione alle materie
    emissionTime = getCurrentTimeUsingCalendar();
    for(MateriaPlus materia : materie)
        materia.setEmissionTime(emissionTime);

    //setto il tempo di vita di ogni materia
    for(MateriaPlus materia : materie)
        materia.setTimeToLiveMinutes(MATERIA_TIME_TO_LIVE_MINUTES);

    //aggiorna le materie solo se nessun altro client le sta leggendo
    MasterServer.readWriteLock.writeLock().lock();

    MasterServer.materiePlus = materie;

    //unlock
    MasterServer.readWriteLock.writeLock().unlock();

    //scrivo su file .json le materie
    materiePlusJson = gson.toJson(materie);
    buffWrite = new BufferedWriter(new FileWriter(recoverFile));
    buffWrite.write(materiePlusJson);
    buffWrite.close();

    for(MateriaPlus materia : materie)
        System.out.println("Json Materia generated:\n" + materia.toString());

    //fatto tutto, si mette in attesa
    Thread.sleep( millis: 1000 * 60 * MateriaPlusUpdater.MATERIA_TIME_TO_LIVE_MINUTES);
}
```

# Lato CLIENT

Server
Tolleranza ai guasti
Highlight Codice
<b>Client</b>
Splash Screen

I client eseguono richieste al server tramite una classe di tipo AsyncTask. Nel metodo 'doInBackground' viene richiesta un'istanza di una classe (pattern singleton) il cui compito è:

- Stabilire una connessione con il server
- Inviare il tipo di laurea del client
- Convertire in 'MateriaPlus' la materia ricevuta in formato Json, e ritornarla

# VotoAsynkTask

Server
Tolleranza ai guasti
Highlight Codice
<b>Client</b>
Splash Screen

```
@Override
protected MateriaPlus doInBackground(Context... contexts) {

    Context context = contexts[0];
    try {
        materiaPlus = ServerCaller.getInstance().getVotoFromServer(laurea);
    } catch (IOException e) {
        String errorMessage = "Unable to reach the server";
        e.printStackTrace();
        Toast.makeText(context, errorMessage, Toast.LENGTH_SHORT).show();
    }

    //System.out.println("MATERIA OTTENUTA:\n\n\n"+materiaPlus.toString()+"\n\n\n");
    return materiaPlus;
}
```

# ServerCaller

Server

Tolleranza ai guasti

Highlight Codice

Client

Splash Screen

```
public MateriaPlus getVotoFromServer(String laurea) throws IOException {  
  
    InputStream input;  
    OutputStream output;  
  
    DataInputStream dataRead;  
    DataOutputStream dataWrite;  
  
    String votoJson;  
  
    Gson gson = new Gson();  
    MateriaPlus materiaPlus;  
  
    Socket socket = new Socket(SERVER_IP, SERVER_PORT);  
  
    input = socket.getInputStream();  
    output = socket.getOutputStream();  
  
    //faccio sapere al client il tipo di laurea che possiede l'utente  
    dataWrite = new DataOutputStream(output);  
  
    dataWrite.writeUTF(laurea);  
    dataWrite.flush();  
  
    //assumo che l'invio dei dati avvenga in modo corretto,  
    //mi fido del controllo degli errori del livello di collegamento  
    //e di quello di TCP  
    dataRead = new DataInputStream(input);  
  
    //ricevo il voto (in formato json) dal server  
    votoJson = dataRead.readUTF();  
  
    //i dati saranno contenuti in un unica linea  
    materiaPlus = gson.fromJson(votoJson, MateriaPlus.class);  
  
    dataRead.close();  
  
    //chiudo la connessione  
    socket.close();  
  
    //ok funge  
    //System.out.println("SERVER CALLER RECIVED: \n\n\n"+votoJson+"\n\n\n");  
  
    return materiaPlus;  
}
```

# RrecyclerView / CardView

Server
Tolleranza ai guasti
Highlight Codice
Client
Splash Screen

All'avvio dell'applicazione la prima activity ad essere aperta sarà la SplashScreen, all'interno della quale l'utente vedrà (nel peggiore dei casi) una rotella di caricamento e delle TextView con lo stato delle operazioni eseguite in tale activity:

- Eventuale creazione del DB
- Creazione dei file SharedPreferences

Per alleggerire il carico di lavoro sul main thread, ognuna di tale operazione è eseguita in parallelo al main thread da delle appropriate AsyncTask.



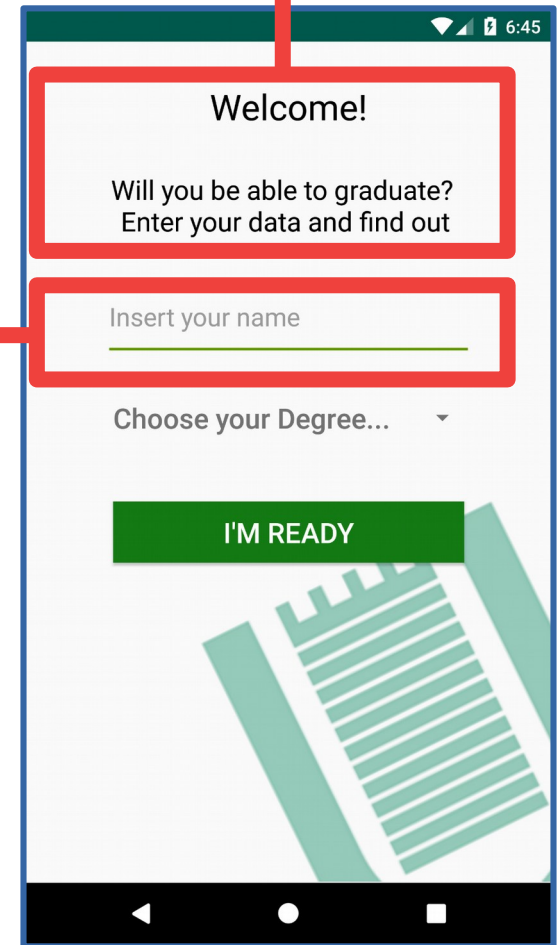
# La EntryPointActivity

La EntryPointActivity viene mostrata al primo avvio dell'applicazione ed è necessaria perché l'utente possa "registrarsi" all'interno dell'applicazione e quindi permettere successivamente al sistema di mostrargli gli esami opportuni. Non verrà più mostrata dopo la compilazione

L'Activity è formata da:

Una EditText dove l'utente inserirà il suo nome

Due TextView di presentazione



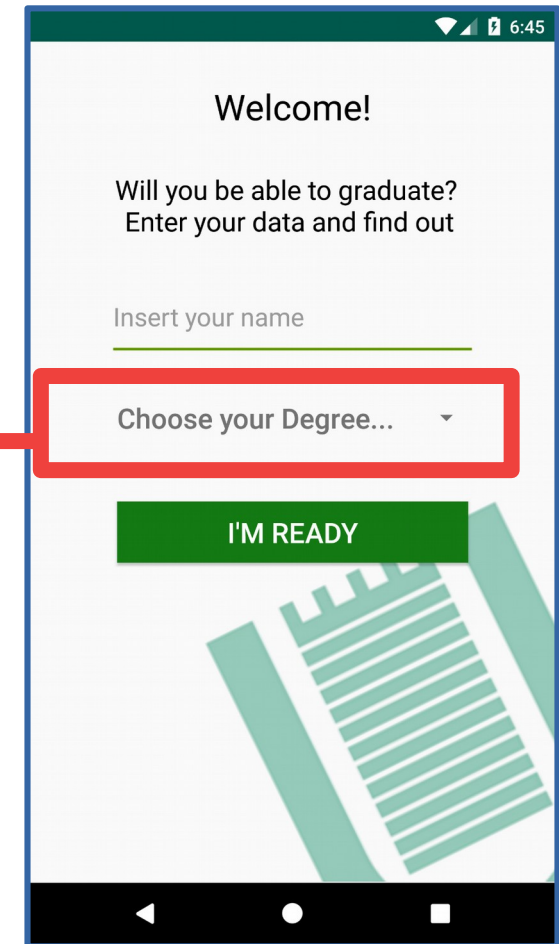
# La EntryPointActivity

La EntryPointActivity viene mostrata al primo avvio dell'applicazione ed è necessaria perché l'utente possa "registrarsi" all'interno dell'applicazione e quindi permettere successivamente al sistema di mostrargli gli esami opportuni. Non verrà più mostrata dopo la compilazione

L'Activity è formata da:

Una EditText dove l'utente inserirà il suo nome

Uno spinner che l'utente utilizzerà per scegliere il corso di laurea preferito



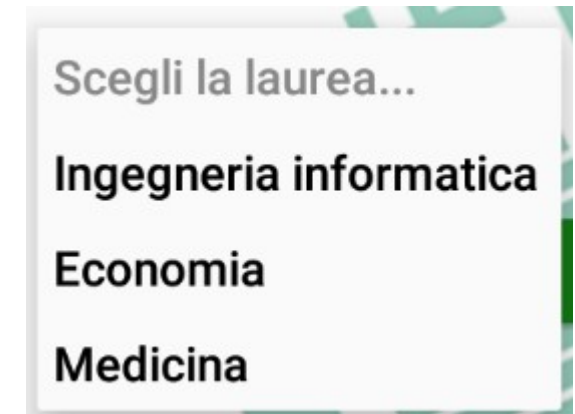
# Lo Spinner dell'EntryPointActivity

Lo spinner implementa al suo interno un adapter del tipo ArrayAdapter, che prende in input un Array (nel nostro caso salvato nel file strings.xml) e associa una TextView ad ogni suo elemento.

Poi abbiamo disabilitato e schiarito il primo elemento così che l'utente possa sempre vedere la richiesta ma senza poterla cliccare

```
// Inizializzo un ArrayAdapter con le lauree
spinnerArrayAdapter = new ArrayAdapter<String>(
    context: this, R.layout.spinner_item, plantsList){
    @Override
    public boolean isEnabled(int position){
        if(position == 0 )
        {
            // Il primo Item lo disabilito per farlo hint
            return false; }
        else
        {
            return true; } }
    @Override
    //setto la tendina
    public View getDropDownView(int position, View convertView,
                                ViewGroup parent) {
        View view = super.getDropDownView(position, convertView, parent);
        TextView tv = (TextView) view;
        if(position == 0 ){
            tv.setTextColor(Color.GRAY);
        }
        else {
            tv.setTextColor(Color.BLACK);
        }
        return view;
    }
}
```

```
<string-array name="Degree">
    <item>Choose your Degree...</item>
    <item>Ingegneria informatica</item>
    <item>Economia</item>
    <item>Medicina</item>
</string-array>
```



# Lo Spinner dell'EntryPointActivity

Nel momento in cui l'utente sceglie il corso di laurea la textView dello spinner verrà annerita e l'informazione inizializzata in un'apposita variabile in attesa di essere salvata in memoria

Ingegneria informatica ▼

```
spinner.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
        String selectedItemText = (String) parent.getItemAtPosition(position);  
  
        if(position > 0){  
            tvSpinner = findViewById(R.id.tvSpinner);  
  
            laurea = selectedItemText;  
            tvSpinner.setTextColor(Color.BLACK);  
        }  
    }  
})
```

# La EntryPointActivity

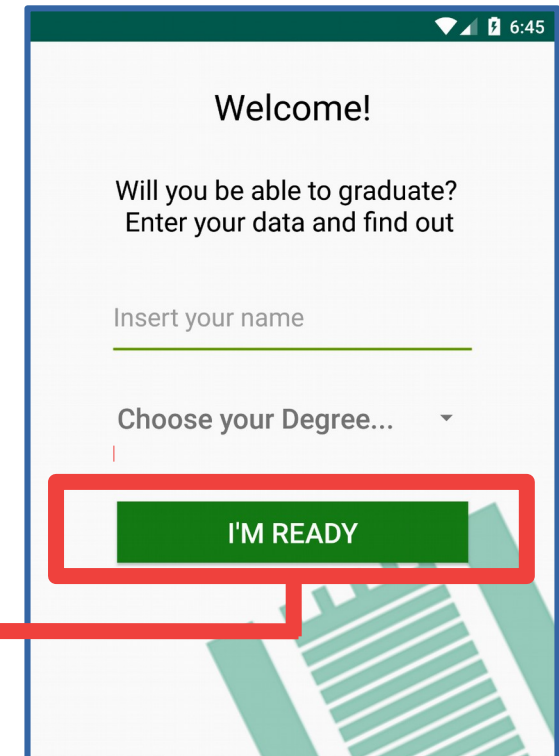
La EntryPointActivity viene mostrata al primo avvio dell'applicazione ed è necessaria perché l'utente possa "registrarsi" all'interno dell'applicazione e quindi permettere successivamente al sistema di mostrargli gli esami opportuni. Non verrà più mostrata dopo la compilazione

L'Activity è formata da:

Una EditText dove l'utente inserirà il suo nome

Uno spinner che l'utente utilizzerà per scegliere il corso di laurea preferito

Un bottone che reagirà solo se è stato inserito il nome e scelto il corso di laurea e salverà questi dati nelle sharedPreferences



```
if (laurea != null && nomeStudiante.getText() != null
    && nomeStudiante.getText().length() > 0 ){

    //Salvo nelle Shared il nome...
    SharedPreferences shared = getSharedPreferences( name: "userData", MODE_PRIVATE);
    SharedManager manager = new SharedManager(shared);

    manager.setUserDataInfo(nomeStudiante.getText().toString(), SharedManager.userDataKey);
}
```

iCarramba DT

# La EntryPointActivity

La EntryPointActivity viene mostrata al primo avvio dell'applicazione ed è necessaria perché l'utente possa "registrarsi" all'interno dell'applicazione e quindi permettere successivamente al sistema di mostrargli gli esami opportuni. Non verrà più mostrata dopo la compilazione

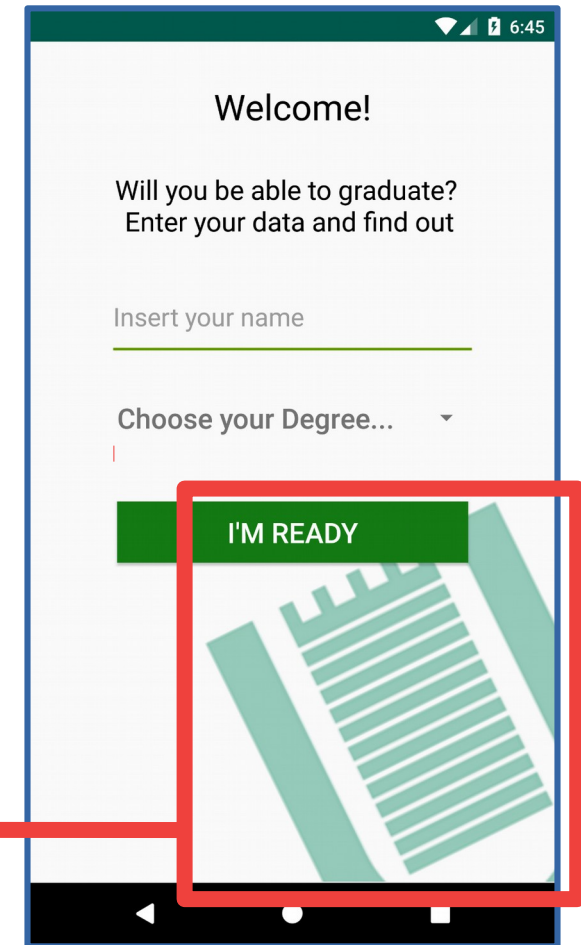
L'Activity è formata da:

- Una EditText dove l'utente inserirà il suo nome

- Uno spinner che l'utente utilizzerà per scegliere il corso di laurea preferito

- Un bottone che reagirà solo se è stato inserito il nome e scelto il corso di laurea e salverà questi dati nelle sharedPreferences

- Una ImageView ancorata nell'angolo resa più trasparente dal metodo della ImageView setAlpha(x)

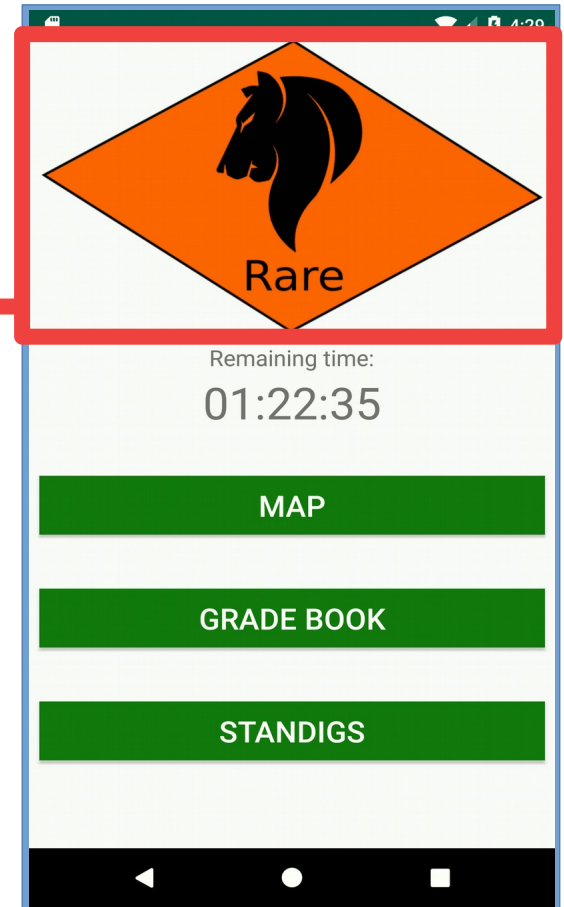


# La MainActivity

La MainActivity è la prima activity che viene mostrata, dopo l'Entry Point la prima volta, e permette all'utente di accedere al resto dell'applicazione e di avere informazioni sulla materia attualmente attiva.

L'Activity è formata da:

Una RarityImageView dove viene mostrata la rarità dell'esame.





# La RarityImageView

La RarityImageView è un'estensione della classe ImageView intesa per contenere solo le immagini della rarità dell'esame.

Ciò viene realizzato con questa semplice funzione che scambia le immagini di rarità:

```
public void changeRarity(int rarity) {  
    switch (rarity) {  
        case 1:  
            //Cambiare immagine a comune  
            this.setImageDrawable(getResources().getDrawable(R.drawable.common,myContext.getTheme()));  
            break;  
        case 2:  
            //Cambiare immagine a rara  
            this.setImageDrawable(getResources().getDrawable(R.drawable.rare,myContext.getTheme()));  
            break;  
        case 3:  
            //Cambiare immagine a mitica  
            this.setImageDrawable(getResources().getDrawable(R.drawable.mythic,myContext.getTheme()));  
            break;  
        case 4:  
            //Cambiare immagine a epica  
            this.setImageDrawable(getResources().getDrawable(R.drawable.epico,myContext.getTheme()));  
            break;  
        case 5:  
            //Cambiare immagine a leggendaria  
            this.setImageDrawable(getResources().getDrawable(R.drawable.legendary,myContext.getTheme()));  
            break;  
    }  
}
```



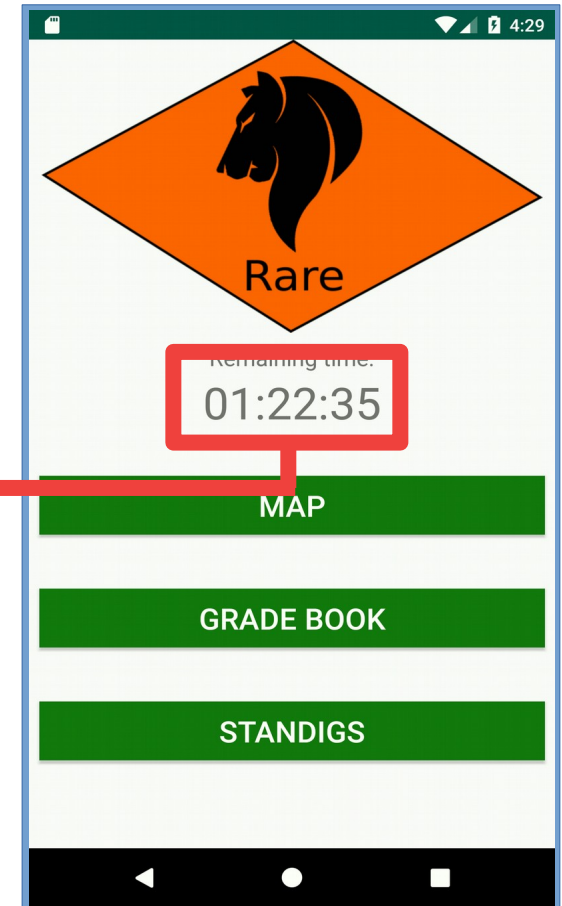
# La MainActivity

La MainActivity è la prima activity che viene mostrata, dopo l'Entry Point la prima volta, e permette all'utente di accedere al resto dell'applicazione e di avere informazioni sulla materia attualmente attiva.

L'Activity è formata da:

Una RarityImageView dove viene mostrata la rarità dell'esame.

Una TimerTextView che mostra il tempo di vita rimanente all'esame



# La TimerTextView

La TimerTextView è una estensione della TextView creata per mostrare un timer all'utente.

La classe usa un CountdownTimer per tenere traccia del tempo trascorso:

```
//Imposto il timer, mi interessano solo i centisecondi
cd = new CountdownTimer(time, countdownInterval: 10) {

    public void onTick(long millisUntilFinished) { setTime(millisUntilFinished); }

    public void onFinish() { tl.onTimerFinished(); }

}.start();
```

A ogni Tick del clock (10 ms) il contenuto della TextView viene aggiornato.

Quando il timer finisce viene chiamata la classe onTimerFinished() di un Listener pre-impostato.

# La MainActivity

La MainActivity è la prima activity che viene mostrata, dopo l'Entry Point la prima volta, e permette all'utente di accedere al resto dell'applicazione e di avere informazioni sulla materia attualmente attiva.

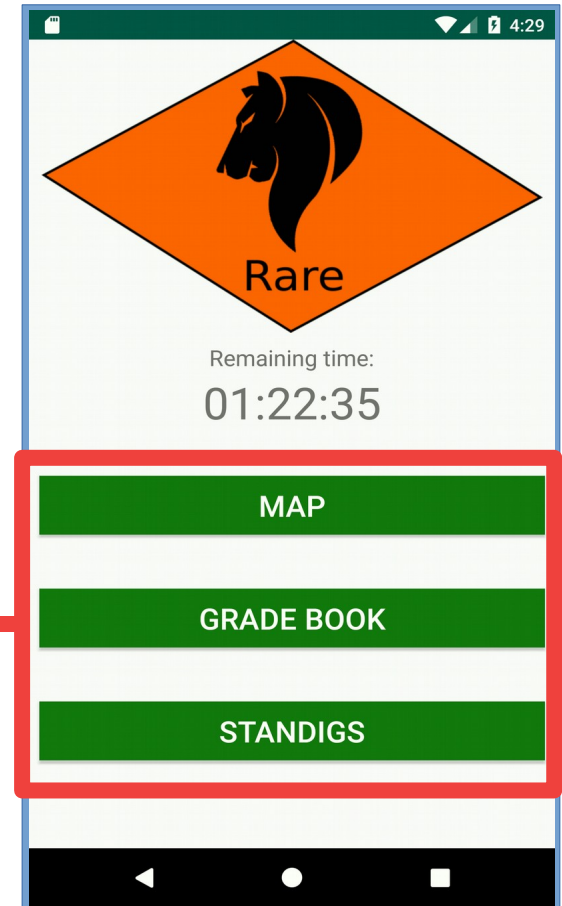
L'Activity è formata da:

Una RarityImageView dove viene mostrata la rarità dell'esame.

Una TimerTextView che mostra il tempo di vita rimanente all'esame

Vari bottoni per raggiungere le altre Activity

Quando l'Activity viene aperta e quando il timer finisce l'Activity usa la VotoAsyncTask per recuperare informazioni sulla materia corrente

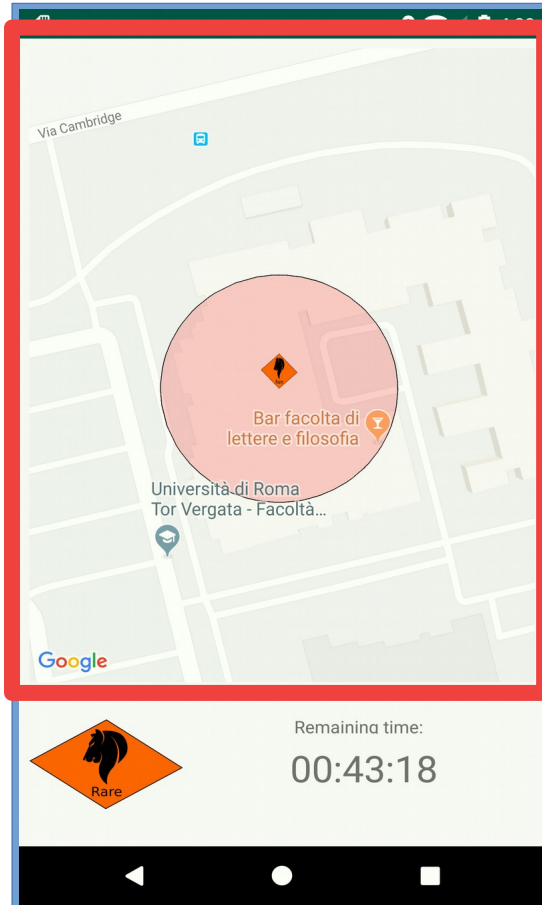


# La mappa

La mappa permette all'utente di capire dove si trova la materia.

L'Activity è formata da:

Una GoogleMap, una View fornita dalle librerie Google dove si vede la posizione dell'utente e un cerchio con l'immagine del voto.



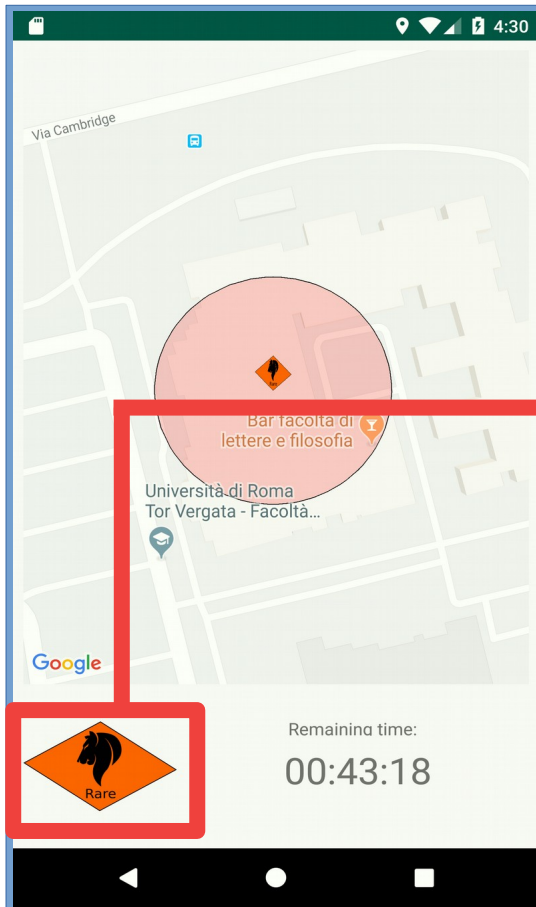
# La mappa

La mappa permette all'utente di capire dove si trova la materia.

L'Activity è formata da:

Una GoogleMap, una View fornita dalle librerie Google dove si vede la posizione dell'utente e un cerchio con l'immagine del voto.

Una RarityImageView, la stessa usata nella MainActivity



# La mappa

La mappa permette all'utente di capire dove si trova la materia.

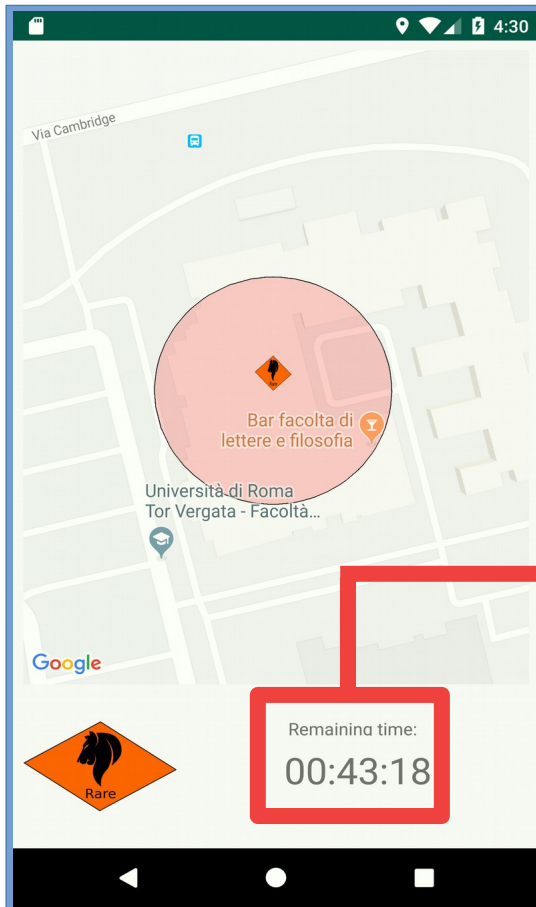
L'Activity è formata da:

Una GoogleMap, una View fornita dalle librerie Google dove si vede la posizione dell'utente e un cerchio con l'immagine del voto.

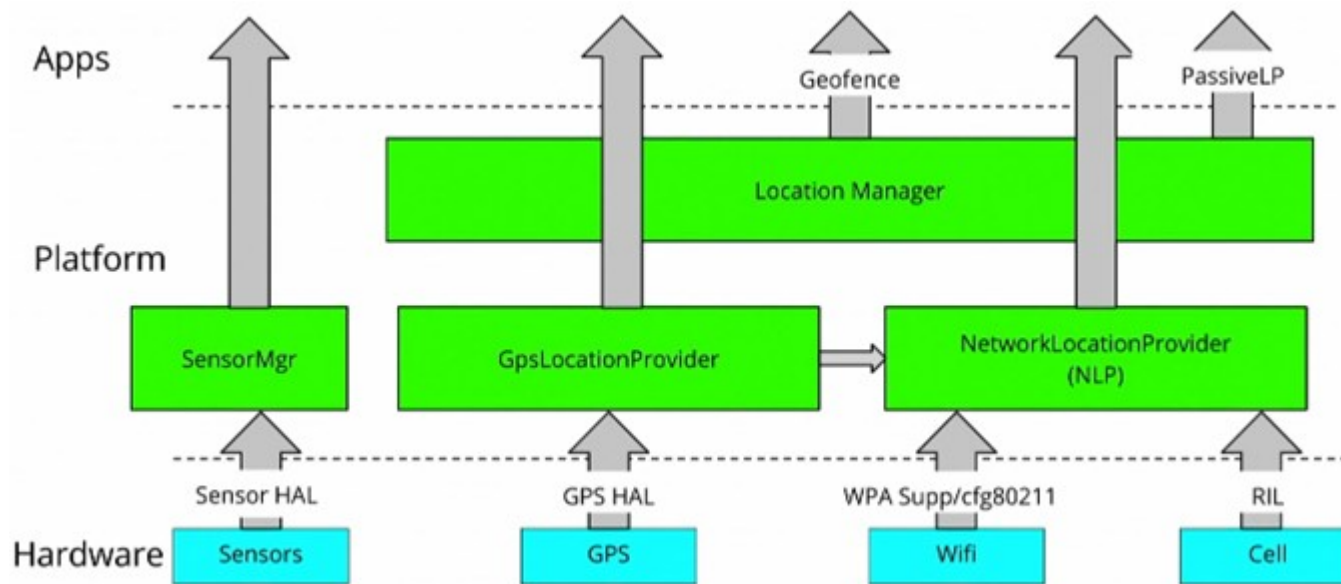
Una RarityImageView, la stessa usata nella MainActivity

Una TimerTextView, la stessa usata nella MainActivity

Anche questa Activity utilizza VotoAsyncTask per recuperare il voto dal Server



# La distanza dell'utente



La distanza dell'utente dal voto viene calcolata usando il LocationManager all'interno dell'Activity con la mappa, dove infatti vengono chiesti per la prima volta i permessi necessari alla geolocalizzazione.

In particolare, ogni volta che arriva una nuova posizione viene calcolata la distanza con il voto per controllare se sia il caso di catturarla.

Per capire quali posizioni sono utilizzabili usiamo il seguente modello:

# Il modello per la geolocalizzazione

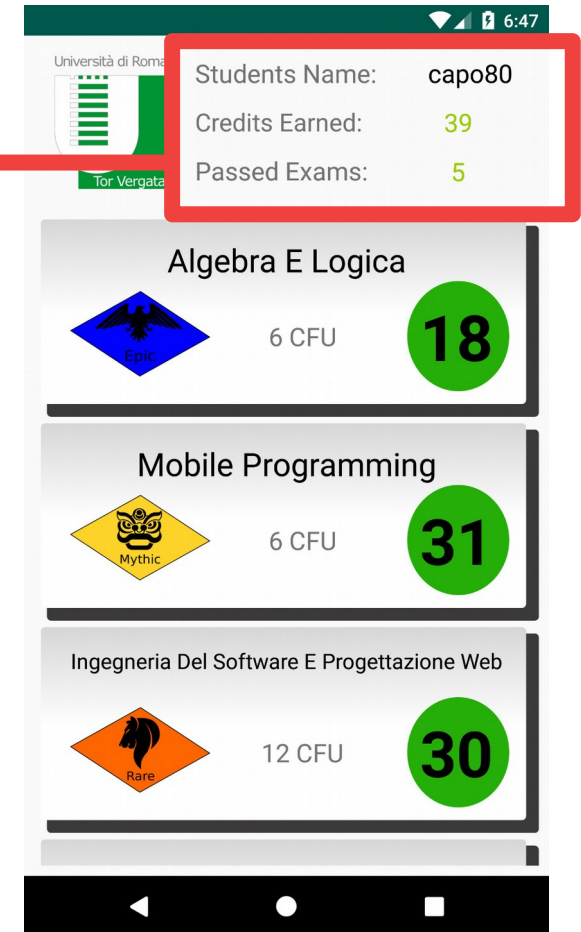
```
private boolean isBetterLocation(Location location, Location currentBestLocation) {  
    //se non ho ancora una location prendo la nuova  
    if (currentBestLocation == null) {  
        return true;  
    }  
    //Se è passato un minuto dall'ultima location prendo la nuova  
    //Se la nuova è più vecchia di un minuto mi tengo la vecchio  
    long timeDelta = location.getTime() - currentBestLocation.getTime();  
    boolean isSignificantlyNewer = timeDelta > ONE_MINUTES;  
    boolean isSignificantlyOlder = timeDelta < -ONE_MINUTES;  
    boolean isNewer = timeDelta > 0;  
  
    if (isSignificantlyNewer) {  
        return true;  
    } else if (isSignificantlyOlder) {  
        return false;  
    }  
  
    // Se la location ha un delta maggiore di 60 metri la scarto  
    int accuracyDelta = (int) (location.getAccuracy() - currentBestLocation.getAccuracy());  
    boolean isLessAccurate = accuracyDelta > 0;  
    boolean isMoreAccurate = accuracyDelta < 0;  
    boolean isSignificantlyLessAccurate = accuracyDelta > 60;  
  
    //Controllo se hanno lo stesso provider  
    boolean isFromSameProvider = isSameProvider(location.getProvider(),  
        currentBestLocation.getProvider());  
  
    //Decido un base ai parametri se accettarla  
    if (isMoreAccurate) {  
        return true;  
    } else if (isNewer && !isLessAccurate) {  
        return true;  
    } else if (isNewer && !isSignificantlyLessAccurate && isFromSameProvider) {  
        return true;  
    }  
    return false;  
}
```



# Libretto

La visualizzazione delle materie ottenute avviene nell'activity 'BookletActivity', contenete:

Delle TextView con informazioni sui crediti e materie complessivamente ottenute dall'utente



# Libretto

La visualizzazione delle materie ottenute avviene nell'activity 'BookletActivity', contenete:

Delle TextView con informazioni sui crediti e materie complessivamente ottenute dall'utente

Una ImageView che verrà modificata nel momento in cui l'utente raggiungerà la laurea



# LIBRETTO

La visualizzazione delle materie ottenute avviene nell'activity 'BookletActivity', contenete:

Delle TextView con informazioni sui crediti e materie complessivamente ottenute dall'utente

Una ImageView che verrà modificata nel momento in cui l'utente raggiungerà la laurea

Una RecyclerView all'interno della quale sono organizzate le informazioni di ciascuna materia ottenuta



# Visualizzazione Item

L'Adapter della RecyclerView per poter essere istanziata necessita di un vettore di 'Materie', i cui attributi verranno estratti e utilizzati per riempire di informazioni ciascun item definito nel layout 'booklet\_item'.

Il ViewHolder dell'Adapter conterrà tutte le View che comporranno gli item del layout che verranno poi 'Gonfiati' dal LayoutInflater.

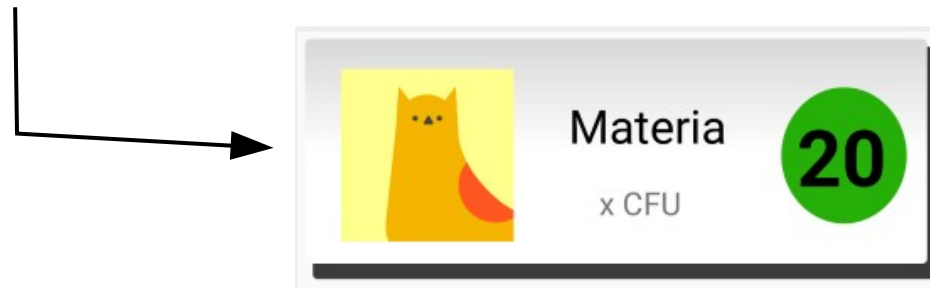
# Visualizzazione Item

```
public static class MyViewHolder extends RecyclerView.ViewHolder {  
  
    // each data item is just a string in this case  
    public MyTextView cfu;  
    public MyTextView materia;  
    public MyTextView voto;  
    public RarityImageView rarImm;  
  
    public MyViewHolder(View v) {  
        super(v);  
        voto = v.findViewById(R.id.tvVoto);  
        materia = v.findViewById(R.id.tvMateria);  
        cfu = v.findViewById(R.id.tvCfu);  
        rarImm = v.findViewById(R.id.iwRarity);  
    }  
}
```

1. Definito 'container' di Views

```
@Override  
public void onBindViewHolder(MyViewHolder myViewHolder, int i) {  
  
    myViewHolder.materia.setText(voti.get(i).getSubject());  
    myViewHolder.voto.setText(Integer.toString(voti.get(i).getMark()));  
    myViewHolder.cfu.setText(Integer.toString(voti.get(i).getCredits()) + " CFU");  
    myViewHolder.rarImm.changeRarity(voti.get(i).getRarity());  
}
```

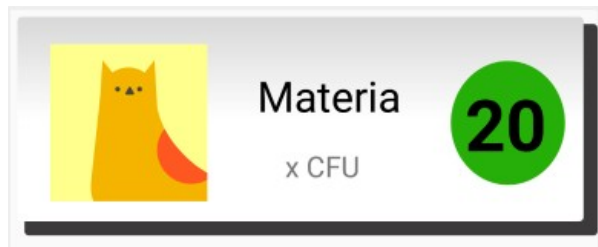
2. Ogni View del 'container' è settata in base agli attributi della Materia



3. Item finale dopo che è stato 'gonfiato'

# RrecyclerView / CardView

Pe dare un aspetto più accattivante a ciascun item del RecyclerView, il ConstraintLayout del booklet\_item ha come sfondo (background) un drawable che simula un effetto 'CardView'.



L'oggetto di tipo drawable è costituito da due item sovrapposti e sfasati. Per il colore di quello in primo è stato usato l'effetto 'gradient'.


```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:top="10dp" android:left="10dp"
        android:drawable="@drawable/plate_shadow"
        android:gravity="fill"/>
    <item android:top="-5dp" android:left="-5dp"
        android:drawable="@drawable/plate"
        android:gravity="fill"/>
</layer-list>
```

# La Classifica

In questa Activity vengono mostrati gli esami catturati ordinati per tempo di cattura e separati per rarità

L'Activity è formata da:

Una PageView, una custom view che mostra le diverse rarità e permette di scegliere tra le diverse classifiche



1.	Ingegneria Del Software E Progettazione Web	30	00:00
2.	Basi Di Dati	18	00:01
3.	Mobile Programming	31	00:09
4.	Algebra E Logica	18	00:12
5.	Automi E Linguaggi	24	00:50
6.	Analisi 1	31	01:32
7.	Fondamenti di elettronica	28	02:10
8.	Sistemi operativi	18	15:12



# La PageView



La PageView è una custom view che implementa menù customizzabile con immagini.

La view accetta I seguenti parametri xml (in caso non fossero inseriti vengono usati dei valori di default):

- PAGE\_NBR = 6, numero di icone mostrate
- VertSpace = 30, spazio verticale tra l'immagine e il bordo del rettangolo
- HoriSpace = 10, spazio verticale tra l'immagine e il bordo del rettangolo
- VertCurve = 20, altezza dell'arrotondamento del rettangolo
- HoriCurve = 10, larghezza dell'arrotondamento del rettangolo
- BackColor = "#787878", colore dei rettangoli più scuri
- foreColor = "#ffffff", colore dei rettangoli più chiari



# La PageView



La PageView utilizza i metodi della Canvas per disegnare i rettangoli e gli archi, mantenendo un valore corrispondente alla pagina attualmente selezionata.

Quando l'utente tocca la View, essa calcola in quale rettangolo è avvenuto il tocco e aggiorna la View se necessario.

Inoltre ad ogni aggiornamento viene chiamato il metodo di un Listener a cui viene comunicato la nuova pagina selezionata.

```
case MotionEvent.ACTION_DOWN:
    float rectSize = (float)wid/PAGE_NBR;
    int prev = currPage;
    currPage = (int)(event.getX()/rectSize);
    if (currPage != prev) {
        invalidate();
        if (pl != null)
            pl.onPageChanged(currPage);
    }
    break;
```

# La Classifica


In questa Activity vengono mostrati gli esami catturati ordinati per tempo di cattura e separati per rarità

L'Activity è formata da:

Una PageView, una custom view che mostra le diverse rarità e permette di scegliere tra le diverse calssifiche.

Una ReceiclerView, simile a quella utilizzata nel libretto universitario, solo con un nuovo layout e con una diversa query nel DB per recuperare gli esami nel corretto ordine.

Questa View non ha bisogno del Server per funzionare ed è quindi accessibile anche quando il dispositivo è offline.



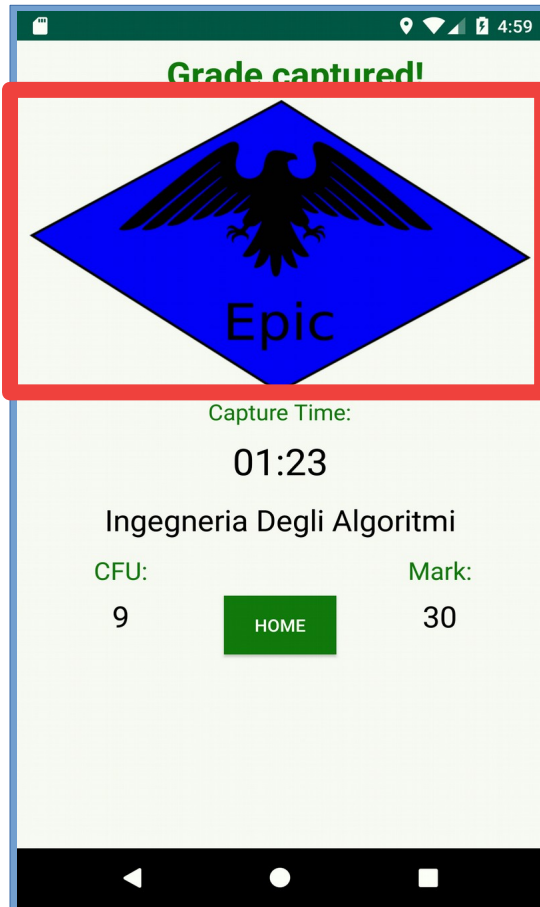
1.	Ingegneria Del Software E Progettazione Web	30	00:00
2.	Basi Di Dati	18	00:01
3.	Mobile Programming	31	00:09
4.	Algebra E Logica	18	00:12
5.	Automi E Linguaggi	24	00:50
6.	Analisi 1	31	01:32
7.	Fondamenti di elettronica	28	02:10
8.	Sistemi operativi	18	15:12

# La cattura del voto

Quando un voto viene catturato viene mostrata questa Activity.

L'Activity è formata da:

Una RarityImageView, la stessa della MainActivity.



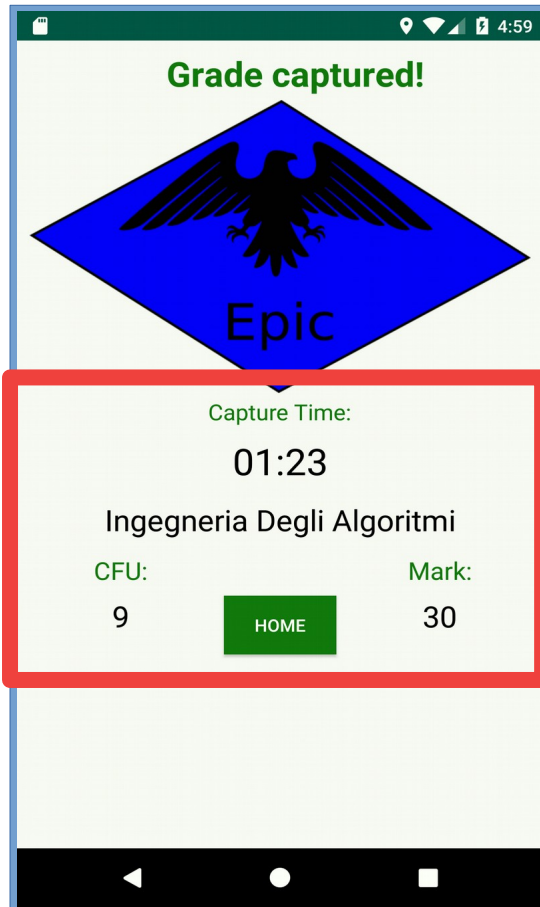
# La cattura del voto

Quando un voto viene catturato viene mostrata questa Activity.

L'Activity è formata da:

Una RarityImageView, la stessa della MainActivity.

Varie Informazioni sul voto catturato e un bottone che porta alla Main Activity



# Notifiche

Innanzitutto per avviare il servizio di sistema riguardante le notifiche, dall'API 26, è necessario creare un canale dedicato

```
private void createNotificationChannel() {  
  
    // Create the NotificationChannel, but only on API 26+ because  
    // the NotificationChannel class is new and not in the support library  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        CharSequence name = "Channel notification";  
        String description = "This is a channel for notifications";  
  
        int importance = NotificationManager.IMPORTANCE_DEFAULT;  
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name, importance);  
        channel.setDescription(description);  
        // Register the channel with the system; you can't change the importance  
        // or other notification behaviors after this  
        NotificationManager notificationManager = getSystemService(NotificationManager.class);  
        notificationManager.createNotificationChannel(channel);  
    }  
}
```

Il servizio di ricezione delle notifiche viene avviato/resettato ogni qualvolta la MainActivity contatta con successo il server attraverso la chiamata della funzione scheduleNotification().

```
try {  
    delay = materiaPlus.getTimerInMillis();  
    resetTimer(delay);  
    scheduleNotification(getBaseContext().getApplicationContext(), delay,  
        repeat: materiaPlus.getTimeToLiveMinutes()*60*1000, NotifListner.NOTIFICATION_IL  
    } catch (ParseException e) {  
        //Errore nella ricezione del pacchetto try again
```

# Notifiche

All'interno della funzione viene creato un PendingIntent della classe NotifListner. Dopodiché viene chiuso, se attivo per quel canale, il servizio di Alarm e ne viene avviato un altro con i dati ricevuti dal server

```
public void scheduleNotification(Context context, long delay, long repeat, int notificationId) {  
  
    Intent notificationIntent = new Intent(context, NotifListner.class);  
    PendingIntent pendingIntent = PendingIntent.getBroadcast(context, notificationId, notificationIntent,  
        PendingIntent.FLAG_CANCEL_CURRENT);  
    //tempo rimasto prima della scadenza del voto attuale  
    long futureInMillis = SystemClock.elapsedRealtime() + delay + 5000;  
  
    AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);  
    alarmManager.cancel(pendingIntent);  
    /**Per far ricevere le notifiche anche in DOZE mode ed in risparmio energetico  
    Si può utilizzare la funzione setAndAllowedWhileIdle, ma sarebbe poco efficiente in termini di batteria  
    essendo le nostre notifiche ripetitive e non di fondamentale importanza per l'utente*/  
    alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP, futureInMillis, repeat, pendingIntent);  
}
```

Dopo essere stato avviato, se l'applicazione viene chiusa, il servizio di Alarm si occuperà di contattare ad intervalli regolari la classe NotifListner, predisposta precedentemente, all'interno della quale abbiamo una nuova chiamata al server in AsyncTask. In caso di insuccesso la classe restituisce il controllo e bisogna attendere l'alarm successivo.

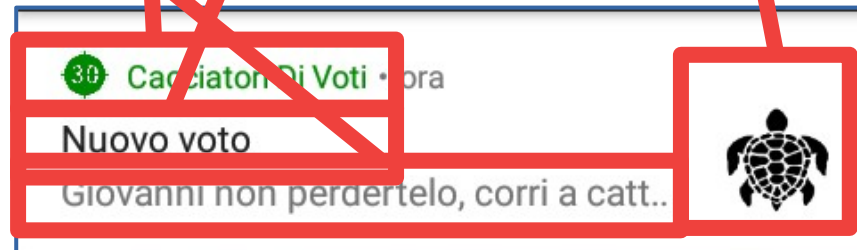


# Notifiche

In caso di successo nel contattare il server viene invece creata la notifica

*//costruisco la notifica*

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
    .setLargeIcon(resizeFunction (context, rarityId))
    .setSmallIcon(R.drawable.logonot)
    .setColor(Color.GREEN)
    .setContentTitle("New Mark!!")
    .setContentText(username + " !!! Don't miss it, run to catch it!!")
    .setSound(Ringtonemanager.getDefaultUri(Ringtonemanager.TYPE_NOTIFICATION))
    .setContentIntent(activity);
```



*//Permette di vedere la notifica anche dal lock screen*

```
builder.setVisibility(Notification.VISIBILITY_PUBLIC);
```

```
NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, builder.build());
```