PROGETTO FINALE MOBILE PROGRAMMING RUN2

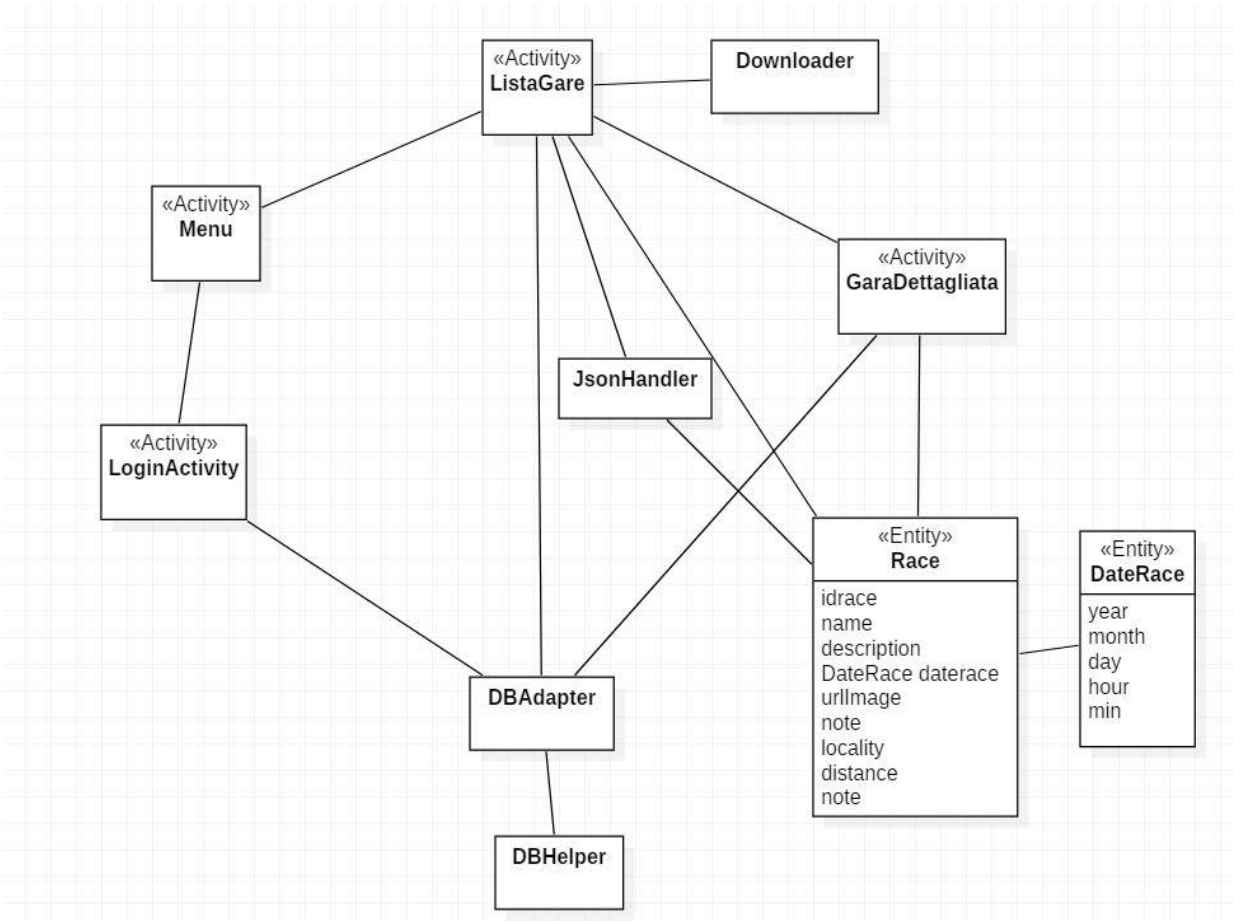
Andrea Di Iorio 0232032

Livio Tiraborrelli 0229096

Manuale di analisi:

- requisiti funzionali:
    - Il sistema dovrà controllare la validità dell'username e password inserita
    - Il sistema dovrà consentire la visualizzazione delle prossime gare
    - Il sistema dovrà visualizzare in una prima fase solo nome, la data, la località e la distanza della gara
    - il sistema dovrà, in seguito a una richiesta dell'utente, visualizzare tutte le informazioni disponibili su una gara
    - il sistema dovrà permettere dove possibile la prenotazione a una gara
    - il sistema dovrà ,in caso di richiesta di prenotazione a una gara , chiedere conferma all'utente di tale decisione prima di effettuare la prenotazione
- requisiti non funzionali:
    - il sistema dovrà scambiare informazioni tramite chiamate di tipo REST a un server(o tramite simulazione di esse).

- Classi di analisi



DESCRIZIONE CLASSI PIU IMPORTANTI

- DBHelper è una classe Helper per la creazione e la gestione del database. Questa classe si occupa di aprire il database se esiste, oppure crearlo se non esiste, e aggiornarlo se è necessario. Ha come attributi le stringhe sql utilizzate per creare le tabelle utilizzate per l'app. Queste ultime vengono create solo se non ancora esistono. L'attributo DATABASE_VERSION è un intero che indica la versione del database: in caso di un suo incremento, tutte le tabelle se esistono vengono eliminate, dopodiché si richiama il metodo onCreate per ricreare le tabelle con delle eventuali modifiche inserite.

- DBAdpter è la classe utilizzata per eseguire query e inserimenti nel database. Utilizza la classe DBHelper per aprire o creare e per ottenere un interfaccia al database. In questa classe sono presenti tutti i metodi utilizzati per eseguire query sul db. È possibile inserire o eliminare un socio, recuperare tutti i soci per utilizzare al meglio l'AutoCompleteTextView del LoginActivity, autenticare un socio in base al username e la password, poi vari metodi per controllare e gestire la prenotazione ad una gara: se è già stata prenotata dall'utente, se sono finiti i posti o se la data di prenotazione è scaduta, e l'inserimento di una prenotazione andata a buon fine. Vi è anche la gestione del caching, in particolare abbiamo inserito delle tabelle per i file scaricati, che vengono salvati e considerati validi, per un eventuale riutilizzo da parte dello stesso utente o un diverso utente, se acceduti nello stesso giorno corrente in cui sono stati scaricati.

- Race è la classe Entity dell'app e contiene tutte le informazioni richieste relative alle corse.

- DateRace: è una entity utilizzata per gestire le date all'interno di questa app. Questa classe è stata creata per consentire di gestire le date all'interno dell'applicazione in modo più utile per gli scopi. Contiene negl'attributi

informazioni temporali fino a una precisione di minuti. Ha un metodo (now) per tornare il l'istante corrente utilizzando la classe java *Calendar*. É possibile anche valutare l'ordine cronologico tra due date.

- Il download dei dati è gestito in Asynctask che notificano asincronamente al main thread dell'activity invocante. DownloaderTask estende Asynctask ed è tipizzato al tipo di ritorno (è utilizzato sia per scaricare json che immagini)

  La logica di download effettiva è delegata alla classe Downloader dove sono presenti metodi specifici per i download che utilizzano le classi di interfacciamento http del package java.net

- JsonHandler è la classe che si occupa del parsing dei json, dove a partire di una stringa di un file json (scaricata o presa da cache) verranno creati delle istanze di Race.
  Il mapping effettivo dei valori sugl'attributi è ottenuto delegando alla classe Race mantendo JsonHandler più generale.

Scelte Progettuali

Le tabelle nel database sono :

- Prenotation (idMember, idRace): contiene l'id del socio e l'id della gara per collegare il socio ad una gara a cui si è prenotato.

- Cache (`url  data_cached , date`): `questa tabella funziona da cache per i dati scaricati. È presente l'url sorgente identificativo di un informazione che è salvata sotto forma di  stringa nella colonna data_cached. Date rappresenta il momento in cui i dati sono stati inseriti nella tabella.`

- User (id ,name, surname , username , password, sex char ,birth_date): contiene tutte le informazioni importanti per gli utenti dell'app.

- Abbiamo scelto di mantenere tutte le info sulle gare in formato json accessibile tramite internet(tramite link a github: https://raw.githubusercontent.com/andysnake96/RUN2SERVER/master/jsonGare.txt)
- La prenotazione di una gara è simulata in un AsyncTask (CheckAvaibilityTask) tramite scrittura sul db, mandando asincronamente i risultati alla activity DetailedRace.
- Abbiamo scelto di fare caching di tutto ciò che viene scaricato con successo nella tabella CACHE database(prevedendo un estensione dell' app)
- Abbiamo scelto di mantenere validi i dati nella tabella CACHE se e solo se sono acceduti nello stesso giorno del loro download.
- Le immagini (serializzate come stringhe) vengono aggiunte alla tabella CACHE

DESCRIZIONE GENERALE

- **Gestione dei Json**

Tutte le informazioni relative alle gare sono racchiuse in un file json mantenuto su un server (github raw repo)

Il parsing del json è effettuato tramite le classi di gestione dei json del package or.json.JSON... per motivi di compatibilità (l'ottima classe Gson non era compatibile con un nostro dispositivo).

Per ottenere una flessibilità a un ipotetico cambiamento delle informazioni contenute nella Entity race è stato utilizzato il seguente approccio:

1. Tramite la classe JSON vengono estratte coppie chiave valore relative agl'elementi del json .
2. Successivamente tramite un costruttore specifico della Entity Race `public Race(List<String> keys, List<Object> values`
3. Iterando sulle coppie chiave – valore tramite il metodo *setAttributeFromMapping(key,value)*
4. vengono mappati i valori (di values) -> sugl'attributi della istanza di Race attraverso le chiavi (keys)

(in questa maniera la classe JsonHandler mantiene una validità abbastanza generica mentre il mapping effettivo delle informazioni sull'istanza è delegato alla classe Entity Race sostanzialmente in due metodi)

- **Download dei dati:**

il json con le informazioni delle gare (contenente anche link a immagini relative alle gare) viene scaricato (o preso dalla tabella CACHE) e successivamente vengono scaricate (o prese dalla tabella CACHE) le immagini tramite la classe Downloader.

Per ogni download viene utilizzato un AsyncTask che notificherà il risultato dell'operazione tramite un metodo di callBack mantenendo il main thread libero (i dati vengono anche aggiunti alla tabella cache tramite DbAdapter)

Una volta ottenuto e parsato il json è possibile calcolare il numero di download da effettuare confrontando i link sul json e le immagini valide presenti nella cache, per dare maggiore interattività di questa operazioni è stata aggiunta una barra di progresso per notificare a che punto è giunta la fase di download.

- **CACHING**

Nell'applicazione viene mantenuta una cache tramite una tabella nel db CACHE(url,data,date) che utilizza come chiave primaria l'url del dato (data) nel web, e mantiene in date la data di download del dato

- Le informazioni vengono considerate valide se accedute all'interno dello stesso giorno del download, inoltre nel caso il json fosse non fosse valido le immagini serializzate nella cache verranno invalidate (in un ottica dove le immagini sono un corredo delle informazioni sulle gare)
- Il metodo di accesso alle informazioni tramite url valuta la validità tramite la colonna date dei dati, in caso troppo vecchia, la riga viene cancellata dalla tabella
- Le immagini vengono serializzate in una stringa testuale prima di essere inserite nella cache tramite l'algoritmo base64 che converte dei raggruppamenti di 6 bit di un dato in caratteri di una stringa.
  Questa operazione è wrappata in :

```
String
rawImgAsStr=Base64.encodeToString(rawBytesImage,Base64.DEFAULT);
writeInCache(rawImgAsStr,url);          //save in cache data
```

Activities dell'app

- L'activity LoginActivity è la prima activity dell'app e serve per l'autenticazione. I dati dei soci sono salvati nel db. Quindi una volta inseriti username e password l'app controlla tramite una query utilizzando la classe DBAdapter l'esistenza dei campi inseriti. Dovrà recuperare anche l'id del socio perché servirà per un eventuale prenotazione alla gara. In questa activity è presente un AutoCompleteTextView per inserire l'username: inizialmente tutti gli username vengono recuperati dal db per impostare i suggerimenti. La password si inserisce in un editText. Poi sono presenti, oltre il bottone che esegue l'autenticazione, delle view che visualizzano eventuali errori, per esempio il non inserimento di uno dei due campi oppure il fallimento dell'autenticazione. Poi c'è un UserLoginTask (che estende un AsyncTak)per eseguire un asincrona autenticazione dell'utente: esegue un interrogazione al db per constatare l'esistenza e la correttezza dei campi inseriti.

- L'activity Menu presenta quattro bottoni per scegliere le tre opzioni che il socio può vedere e in più un altro per il log out. In questa activity oltre il log out è attivo il bottone che fa visualizzare le prossime gare a cui il socio potrebbe prenotarsi. Il log out non fa altro che tornare alla pagina iniziale per l'autenticazione

- L'activity listaGare si occupa di visualizzare le gare scaricate (o prese da cache) dopo essere opportunamente trattate. Alla pressione di una gara si andrà a vedere le info della gara in maggior dettaglio in GaraDettagliata.

- L'activity GaraDettagliata visualizza in dettaglio le informazioni della gara richiesta. Se possibile sarà presente il bottone conferma per prenotarsi alla gara o un messaggio che spiega l'impossibilità della prenotazione.