Esame di stato A.S. 2015/2016

Schoolendar

Zanelli Gabriele
Classe 5 Sezione AI
IIS “Benedetto Castelli”

Indice

Introduzione

Descrizione

Schoolendar è un'applicazione **nativa** per sistema operativo Android nata come strumento per la gestione degli impegni scolastici in utilizzo alternativo ad un normale calendario poiché più approfondita e mirata all'ambito scolastico e sviluppata per la personalizzazione degli impegni a seconda delle diverse esigenze.
Ispirata ad un'applicazione di successo esistente per la piattaforma iOS chiamata **iStudiez**, che tuttavia non offre una soluzione in ambito Android.

Applicazioni native vs Web apps

Un'app nativa è un'applicazione sviluppata con codice e libreria proprietarie, che viene scaricata e installata sul dispositivo, i cui principali vantaggi sono:

- poter interagire con la maggior parte delle **features** del dispositivo (ad esempio accedere alla rubrica, ai messaggi, alla galleria...);
- **esecuzione** diretta sull'hardware del dispositivo, ottimizzando le prestazioni e l'esperienza utente;
- funzionamento **offline**;
- essere pubblicata sullo **Store** corrispondente, accedendo a possibilità di promozione e pubblicizzazione da parte di quest'ultimo.

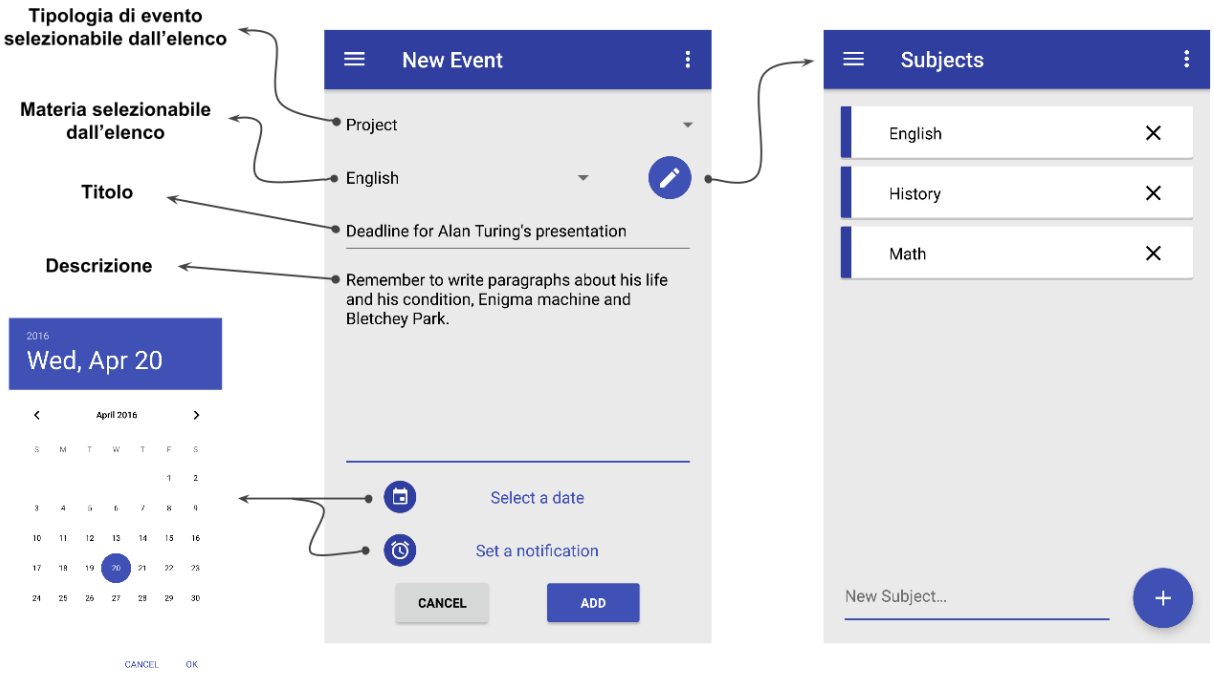
Una web app è un'applicazione accessibile da un qualsiasi browser che supporti le tecnologie utilizzate per svilupparla, i cui principali vantaggi sono:

- non richiedere alcuna **installazione** per l'esecuzione , quindi anche nessun eventuale aggiornamento;
- piena **compatibilità** con ogni ambiente, essendo eseguita su un browser non ha bisogno di essere riscritta per ogni piattaforma;
- **versatilità** nello sviluppo, infatti a differenza delle app native, scritte nel linguaggio di programmazione definito per la piattaforma, le web app possono essere sviluppate con qualsiasi tecnologia si ritenga migliore.

Funzionalità

Creazione di un evento

Lo scopo principale dell'applicazione è consentire all'utente la creazione di eventi personalizzabili nel contenuto per tipologia e per materia, in modo da garantire chiarezza nell'organizzazione dei propri impegni, impostarne una scadenza ed un eventuale avviso di notifica.



Sincronizzazione tramite Cloud

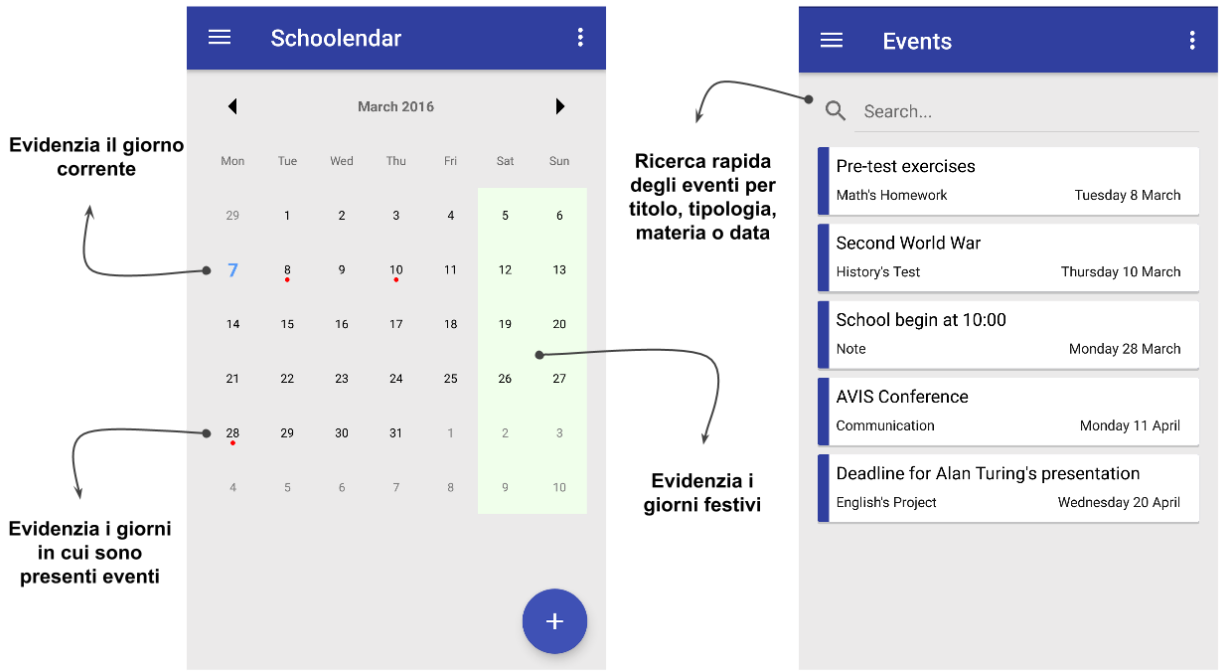
L'utente può registrarsi facoltativamente tramite un indirizzo email ed una password oppure in alternativa può utilizzare il proprio account Google o Facebook come metodo di registrazione, per avvalersi della possibilità di sincronizzare i propri eventi e le preferenze con il Cloud, conservando i propri dati in caso di rimozione dell'applicazione o di utilizzo di un altro dispositivo.

Integrazione con ClasseViva Spaggiari

L'applicazione è in grado, tramite le credenziali di accesso a ClasseViva Spaggiari concesse dall'utente, di effettuare richieste al server originale per raccogliere dati su eventi creati nell'agenda di classe e mostrarli nei propri eventi personali con un tag speciale che ne dichiara la provenienza.

Modalità di visualizzazione

La totalità degli eventi creati dall'utente è visualizzabile in diversi modi, a seconda delle proprie esigenze, infatti questi vengono disposti all'interno di un calendario scorrevole che ne rappresenta una rapida visione mensile, ma possono anche essere visualizzati come una lista filtrabile per tipologia, materia e data per facilitarne la ricerca.



Sviluppo

Android Studio

L'applicazione è stata realizzata utilizzando Android Studio, un ambiente di sviluppo integrato sviluppato da Google, ufficiale e specifico per la piattaforma Android, che utilizza **IntelIJ** come editor e **Gradle** come strumento di build automation.

Linguaggio di programmazione

Il linguaggio di programmazione per la piattaforma Android è Java, mentre per quanto riguarda la definizione della struttura delle applicazioni e del contenuto grafico di ogni schermata, vengono utilizzati dei file XML contenenti i layout.

Java

Java è un linguaggio di programmazione orientato agli oggetti prodotto dalla Sun Microsystems e realizzato da un team di ingegneri con a capo James Gosling, nel 1992.
Le caratteristiche principali del linguaggio sono:

- portabilità del codice attraverso l'utilizzo di una macchina virtuale per l'esecuzione di esso, indipendente dalla piattaforma di esecuzione, proprietà sulla quale è basata l'idea dell'intero linguaggio: dal motto "Write Once, Run Anywhere", "scrivi una volta, esegui ovunque";
- sintassi ereditata da C++ e caratteristiche di orientamento agli oggetti ispirate ad Objective C;
- disponibilità di numerosissime librerie, comprese quelle prodotte da terze parti, per l'implementazione di diverse funzionalità a seconda delle necessità.

XML

XML (eXtensible Markup Language) è un meta-linguaggio di markup utilizzato esclusivamente per la definizione di altri linguaggi, infatti si tratta di un insieme standard di regole sintattiche per modellare la struttura di documenti e dati attraverso una struttura gerarchica composta da:

- componenti determinati da specifici tag denominati **elementi**;
- informazioni che ne descrivono le proprietà denominati **attributi**.

Per far si che un documento XML sia valido rispetto ad un certo linguaggio, è necessario definire la sua "grammatica", specificando i tag utilizzabili e come è possibile strutturarli gerarchicamente.
La **grammatica** di un file XML è definibile tramite linguaggi di definizione tra cui troviamo DTD (Document Type Definition) e XML Schema.
I software che si occupano di verificare se un file XML è ben formato e valido rispetto ad una definizione, sia essa contenuto in un DTD o in uno schema, si chiamano **parser**.

Gradle

Come accennato in precedenza, Android Studio si avvale di Gradle come strumento di build automation, in particolare questo tool offre diverse funzionalità:

- rende semplice la dichiarazione delle dipendenze locali e remote dell'applicazione;
- molto veloce durante il processo di build di un progetto poiché effettua il caching delle build precedenti, evitando la compilazione completa ed effettuando ogni volta build parziali;
- genera APK multipli da uno stesso modulo, per esempio per supportare diverse tipologie di dispositivi o per creare versioni free e pro dell' applicazione;
- automatizzazione dei task ovvero delle singole operazioni da svolgere all'interno delle fasi del procedimento di build;

JUnit

Durante lo sviluppo di un'applicazione per qualsiasi piattaforma, le procedure di testing condotte da operatori umani non risultano quasi mai sufficienti: per ovviare a questo problema è spesso necessario ricorrere a dei **test automatizzati** per assicurarsi che ogni singola unità di sviluppo dell'applicazione svolga correttamente il proprio compito seguendo i requisiti proposti, pratica che viene chiamata **Unit Testing** e Android Studio integra al suo interno JUnit come strumento di test.

Ciò si rende necessario per assicurarsi delle solide basi su cui costruire la propria applicazione evitando errori di tipo logico effettuando i singoli test viene proponendo un input ad un metodo e controllando la bontà di quest'ultimo, ovvero la corrispondenza del risultato ottenuto con l'effettivo risultato richiesto.

NodeJS

Si tratta di un Framework utilizzato per la realizzazione di applicazioni web in linguaggio Javascript, basato su **JavaScript Engine V8**, runtime di Google utilizzato anche dal browser Chrome.
Il vantaggio principale nel suo utilizzo è la modalità di accesso alle risorse orientata agli eventi, lanciando un'azione solo quando viene richiesta e permettendo al sistema operativo di effettuare altre operazioni durante l'attesa, adottando un comportamento **asincrono**.

Git

Come software di controllo di versione è stato utilizzato Git, uno strumento che oltre ad offrire un servizio di versioning del codice, permette la creazione di **branches** rendendo molto più facile la collaborazione in team all'interno di uno stesso progetto nella creazione di **features** occupandosi poi di gestire il merge del codice in modo efficace.
Il software è pensato per la creazione di una repository locale che permette di lavorare su di essa anche **offline**, rendendo poi effettivamente pubbliche le modifiche una volta effettuata la riconnessione alla rete.

GitHub

GitHub è un servizio di hosting web per repository gestite con Git, i cui principali vantaggi nel suo utilizzo combinato a Git sono:

- Il **backup** della propria repository online;
- La rapida **condivisione** del proprio progetto con chiunque;
- La possibilità di rendere pubblica e facilmente reperibile la **documentazione** del proprio codice.

Genymotion

Poiché Android è supportato da miliardi di dispositivi diversi, in fase di debug risulta spesso necessario testare il codice su diversi dispositivi che si differenziano per:

- grandezza dello schermo;
- densità di pixel;
- versione del sistema operativo.

Per rendere questo possibile è stato utilizzato Genymotion, un software che permette di creare illimitati emulatori di dispositivi Android con svariate proprietà.
Inoltre questo emulatore risulta essere molto veloce e ricco di toggle rapidi rispetto al classico emulatore fornito da Android Studio.

Componenti di Android

Android Manifest

L' Android Manifest è un file XML che viene utilizzato per la definizione della struttura dell'applicazione, comprendendo:

- i **Permessi** di cui l'applicazione necessiterà per il proprio corretto funzionamento, che dovranno essere concessi dall'utente;
- le **Activity** create nell'applicazione, ovvero le diverse schermate che verranno visualizzate, determinandone una "di lancio" che verrà avviata al momento del click sull'icona dell'applicazione, specificando per ogni Activity ulteriori proprietà necessarie come filtri per la ricezione di determinati dati;
- i **Service**, ovvero le classi che verranno eseguite in background dall'applicazione, anche quando l'utente non interagirà con essa;
- eventuali **Broadcast Receiveir** di eventi generati dal sistema operativo Android o da altre applicazioni che possono essere raccolti e gestiti da ogni applicazione, specificandone i filtri corrispondenti ad ogni azione.

Activity

La navigazione all'interno di un'applicazione Android si compone di Activity che rappresentano le singole schermate create all'interno di essa per permettere all'utente di interagirvi.
Una nuova Activity è realizzata definendone il contenuto grafico tramite un file XML e associandovi una classe Java che estenda la classe Activity o una delle diverse sottoclassi.

Ogni Activity è dotata di un **Life Cycle** definito dal sistema Android per limitare sprechi di risorse sul dispositivo quando non sono richieste dall'utente.
Questo ciclo vitale è determinato tramite i seguenti metodi autonomamente implementati da ogni Activity e di cui è possibile effettuare l'Override per migliorare l'esperienza di utilizzo dell'applicazione:

- onCreate: chiamato alla creazione dell'Activity, si occupa di inzializzaziarne ogni componente;
- onStart: chiamato successivamente ad onCreate al momento della creazione oppure nel caso in cui l'Activity sia stata precedentemente posta in stato di stop;
- onResume: chiamato successivamente ad onStart oppure nel caso in cui l'Activity sia stata precedentemente posta in stato di pausa per riprenderne l'esecuzione;
- onPause: definisce il comportamento dell'Activity ogni volta che viene nascosta da altri oggetti, come Popup;
- onStop: definisce il comportamento dell'Activity ogni volta che viene spostata in background, ovvero quando l'utente non può più interagire con essa;
- onDestroy: chiamato alla distruzione dell'Activity da parte del sistema Android, questo metodo deve occuparsi di rilasciare tutte le risorse utilizzate dall'Activity.

Layout XML

Ad ogni Activity è necessario associare un file XML contenente il suo layout, ovvero l'insieme dei componenti grafici della schermata e delle loro proprietà, dichiarati tramite specifici tag XML e personalizzabili tramite i corretti attributi.

XML Resources

Oltre ai layout è possibile definire ulteriori file XML il cui contenuto sarà accessibile in ogni parte dell'applicazione, contenenti:

- background personalizzati o altre utilità di tipo grafico definite nella cartella **drawables**;
- animazioni e transizioni applicabili agli oggetti creati nel layout XML definiti nella cartella **anim**;
- template di stili applicabili alle activity nei quali è possibile dichiarare colori di default, proprietà dell'activity e molto altro, definiti nella cartella **styles**;
- valori universali per l'applicazione definiti nella cartella **values** e divisi a loro volta in:
    - Stringhe di testo definite nel file **strings**;
    - Dimensioni, margini, spaziature e grandezze di testo definite nel file **dimen**;
    - Colori in esadecimale, definiti nel file **colors**;
    - Valori di particolare importanza come per esempio eventuali ID, definiti nel file **integers**.

In particolare l'utilizzo di un file XML per le stringhe di testo, permette la traduzione automatica dell'intera applicazione in più lingue semplicemente ridefinendo lo stesso file in tutte le lingue desiderate e ponendo ognuno l'interno di una cartella **values** nel quale viene identificata la lingua, per esempio values-it o values-fr: Android sceglierà automaticamente la cartella da cui prelevare le stringhe basandosi sulla lingua di default del dispositivo.

Intent

Quando è necessario avviare, effettuare operazioni o scambiare dati con altri componenti di Android esterni a quello corrente, per esempio quando è necessario avviare un'Activity, entrano in gioco gli Intent.
Questi sono strumenti di Android in grado di operare a livello di sistema e richiedere l'esecuzione di altri componenti, si dividono in impliciti e espliciti a seconda del loro utilizzo:

- Impliciti: per inviare eventi in broadcast, poiché inviano al sistema Android un evento contenente l'azione che dev'essere svolta, che sarà poi catturata da un Broadcast Receiver.
- Espliciti: per richiedere l'avvio di un'Activity o di un Service, poiché chiamano direttamente il componente di cui hanno bisogno, ovvero la classe Java.

Ogni Intent può portare con sè delle informazioni dalla componente di provenienza a quella di destinazione utilizzando gli **Extras**, ovvero una grande mappa tramite la quale è possibile inviare qualsiasi oggetto serializzabile tramite coppia chiave-oggetto.

Service

Un Service è una componente non dotata di interfaccia grafica che all'interno dell'applicazione Android si occupa di svolgere un determinato compito in **background** che solitamente si identifica in un'operazione che richiede molto tempo per essere portata a termine.
Viene definito tramite una classe Java che estende la classe Service o IntentService, viene avviato da un'altra componente dell'applicazione e una volta terminato il suo compito, viene distrutto.

Fragment

I Fragment possono essere visti come delle porzioni di Activity che hanno come scopo di rendere l'interfaccia **modulare**, ovvero componibile a seconda delle esigenze, per esempio la differenza di spazio a disposizione al ruotare dello schermo.
Il ciclo di vita dei Fragment è strettamente legato a quello dell'Activity di appartenenza, infatti un Fragment non può esistere se non all'interno di un'Activity. Tuttavia si compone di alcuni metodi riguardanti la sua inizializzazione:

- onAttach: chiamato nel momento in cui il Fragment viene assegnato ad un'Activity, in questo momento l'Activity non è stata ancora creata quindi non vi si può interagire;
- onCreate: chiamato alla creazione del Fragment in quanto componente dell'Activity;
- onCreateView: chiamato alla creazione del layout del Fragment;
- onActivityCreated: chiamato una volta completata la creazione dell'Activity, dopo questo momento si potrà interagire con l'Activity.

Broadcast Receiver

Sono componenti che restano in ascolto degli **eventi** inviati dal sistema Android, filtrandoli per la tipologia dichiarata nell'Android Manifest.
Per esempio quando l'utente fa click su un link, viene inviato un Intent Broadcast contenente l'azione di visualizzazione della pagina web, tutte le applicazioni che implementano un Broadcast Reicever filtrato per la visualizzazione di pagine web riceveranno questo evento e, una volta che l'utente avrà deciso con che applicazione portare a termine l'azione, il Broadcast Receiver corrispondente verrà avviato.

ART vs DALVIK

Alla base dell'esecuzione delle applicazioni in Android 5.0 o superiori troviamo ART (Android RunTime) che utilizza un compilatore AOT (ahead-of-time), ovvero che si occupa dell'intera compilazione del codice dell'applicazione durante la sua installazione iniziale e non durante l'esecuzione della stessa.
Il vecchio Dalvik è invece basato sulla tecnologia JIT (just-in-time), ovvero è un compilatore che esegue e compila il codice in linguaggio macchina in tempo reale, ad ogni esecuzione dell'applicazione stessa.
ART inoltre, sfrutta un nuovo garbage collector in grado di gestire la memoria in modo da ridurre il numero e la durata di pause, fornendo così una migliore reattività e un minor utilizzo di risorse.

Gestione dei Dati

Shared Preferences

Le Shared Prefenceres sono strumenti di Android che permettono la memorizzazione di **mappe** di chiavi-valore sotto forma di file XML con cui ogni applicazione Android può interagire, creandone di nuove (pubbliche per tutte le altre applicazioni installate sul dispositivo o private per sè stessa) o recuperando informazioni da quelle già esistenti.
Vengono principalmente utilizzate per salvare poche quantità di dati indicanti informazioni relative all'utente come per esempio un nickname, o il più elevato punteggio ottenuto all'interno di un gioco.

Settings Preferences

Il sistema Android permette la creazione di una schermata di **impostazioni** per la propria applicazione, senza doverne definire il layout, direttamente collegata alle Shared Preferences, definendo un file XML contenente tag riguardanti il nome di ogni impostazione, la **chiave** per il salvataggio nelle Shared Preferences, la tipologia di dato ed un eventuale valore di default.
Questa schermata di impostazioni viene definita estendendo un' ActivityPreference o FragmentPreference e si occupa automaticamente della gestione dei componenti grafici e della sincronizzazione delle impostazioni con la memoria, rendendo estremamente semplice la gestione di queste ultime.

Database

Per la memorizzazione di maggiori quantità di dati è stato necessario l'utilizzo di un database e tra le varie tipologie di database relazionali è stato scelto **SQLite** per i seguenti motivi:

- Estrema portabilità dovuta al fatto che l'intero database si trova all'interno di un singolo file;
- Significativamente più leggero rispetto a MySQL a patto di sacrificare qualche funzionalità come RIGHT OUTER JOIN e FOR EACH;
- Non necessita di alcun server di gestione e vi si può interagire direttamente su filesystem;
- Non implementa la gestione degli utenti, il che lo rende particolarmente adatto all'utilizzo su un dispositivo mobile, che nella maggior parte dei casi è utilizzato dal solo proprietario.

Interazione con il Web Server Spaggiari

Attraverso l'utilizzo di un'applicazione esterna in NodeJS chiamata "ClasseViva API" che effettua richieste al server originale di Spaggiari utilizzando le comuni credenziali di accesso e ritorna i dati richiesti in formato JSON, è possibile interagire facilmente con ClasseViva tramite l'utilizzo di API Rest in grado di recuperare dal server:

- eventi creati nell'agenda di classe;
- voti assegnati, materia e data;
- file caricati dai professori, nome del professore, cartella di appartenenza e relativo link di download.

L'applicazione, utilizzata durante un primo approccio, è stata successivamente rimpiazzata con un'altra soluzione in grado di ottenere un risultato equivalente, è quella di effettuare le richieste HTTP al server Web Spaggiari direttamente dall'applicazione Android, senza l'utilizzo di un server esterno di supporto, il che richiede una minore attesa da parte dell'utente e si identifica in un approccio più diretto e meno costoso in termini di risorse.
Per fare ciò è necessario comunicare con il server di Spaggiari tramite il protocollo HTTP e gestendo le sessioni correnti, richiedere i dati necessari occupandosi di convertire il body HTML ottenuto in dati utilizzabili.

Formato JSON

Per ovviare problemi di memorizzazione di oggetti complessi tramite strumenti come le Shared Preferences, ogni oggetto di tipo non-nativo è stato trasformato in formato JSON e viceversa al recupero di esso utilizzando **GSON**, una libreria sviluppata da Google che semplifica le transizioni Oggetto-JSON e JSON-Oggetto tramite funzioni che accettano come parametro la classe di appartenenza dell'oggetto per la sua ricostruzione.

Autenticazione e Sincronizzazione

Firebase

Per ottenere un semplice servizio di autenticazione tramite credenziali è stato utilizzato un tool di nome Firebase, recentemente acquistato da Google, che tramite l'utilizzo di una console di comando da web permette di registrare la propria applicazione Android/iOS, in modo da gestire in modo semplice un server completo di numerose funzionalità.

Autenticazione

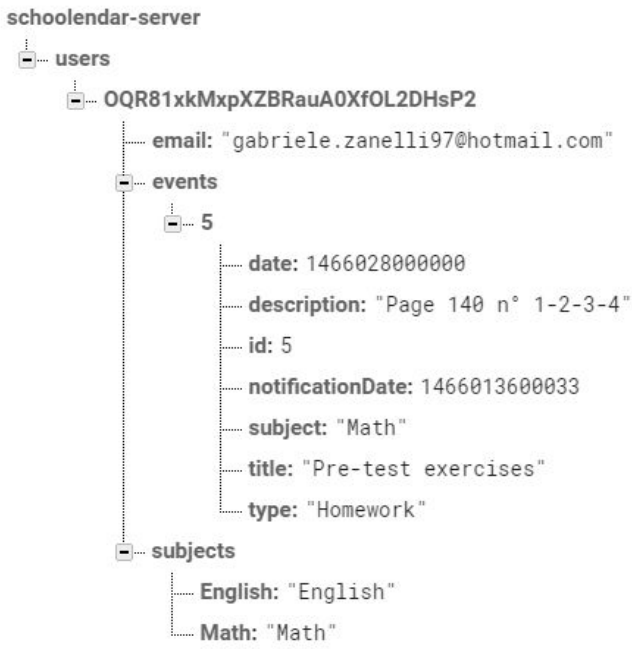
Firebase offre diverse metodologie per l'implementazione dell'autenticazione dell'utente avvalendosi di **provider** di autenticazione esterni come lo stesso Google, Facebook, Twitter etc. oppure permette di registrarsi ed accedere tramite un indirizzo email ed una password, offrendo comunque la possibilità di collegare l'account creato agli altri account forniti dai diversi provider, riconoscendoli come lo stesso utente.

Database Online

Una volta autenticati al server, è possibile accedere all'intero database online (perciò vanno definite delle regole che permettano ad ogni utente di accedere solo ai propri contenuti) per salvare o richiedere informazioni in formato **JSON**.

Infatti tutto il database è strutturato ad albero e ogni parte di esso è accessibile tramite delle **references** definendo il percorso a partire dal nodo root e passando per ogni child.

Il database permette di aggiungere ogni valore tramite l'utilizzo di una chiave di identificazione che ne definisce il percorso, oppure ci si può avvalere della funzionalità **push**, che genera in automatico una chiave univoca basata sul tempo attuale in millisecondi, associandovi il valore desiderato.



Realtime

Punto forte nell'utilizzo del database di Firebase è la possibilità di impostare dei **listener** per numerosi eventi interni al database, restando in ascolto per esempio sull'aggiunta, cambiamento o rimozione di un child all'interno di un determinato ramo del database, ottenendo come risultato l'aggiornamento istantaneo dei dati visualizzati dall'utente.

Firebase Storage

Firebase consente l'accesso ad uno storage, attualmente utilizzato per la memorizzazione delle immagini del profilo dell'utente, suddiviso gerarchicamente allo stesso modo di un filesystem, accessibile tramite **references** che possono puntare a cartelle o a file.
Attraverso una reference è possibile caricare o scaricare byte di dati tramite oggetti in memoria, stream o file nella memoria permanente.
Inoltre ogni reference ad un file contiene dei **meta-data** di default tra cui nome del file, tipologia di contenuto, grandezza, data di creazione etc. ai quali è possibile aggiungerne di nuovi arbitrariamente, personalizzandone i riferimenti.

Implementazione

Modularità

L'applicazione è composta da un'unica Activity principale che si occupa, attraverso un menù di **navigazione**, di gestire le transizioni tra le varie schermate che si identificano in Fragment.
L'approccio modulare che compone l'applicazione permette di ottenere un risultato meno esoso in termini di risorse poichè l'utilizzo di Fragment per la navigazione attiva risulta più performante rispetto all'utilizzo di nuove Activity per ogni schermata.

Visione mensile

La schermata "Month View" è interamente occupato da un componente grafico chiamato **CalendarView**, originario di una libreria esterna, che implementa un calendario mensile scorrevole personalizzabile in modo dinamico tramite istanze di classi Java che estendono **Decorator**, in particolare sono state create tre classi per:

- la rapida visualizzazione dei giorni in cui sono presenti le tipologie di eventi, tramite l'utilizzo marcatori colorati;
- l'highlight dei giorni di fine settimana tramite un diverso sfondo;
- l'highlight del giorno corrente tramite una diversa colorazione e formattazione del testo.

Visione per lista filtrabile

Nella schermata "All Events" sono visualizzati tutti gli eventi futuri in ordine cronologico, sotto forma di **Cards** offrendo la possibilità di effettuare una rapida ricerca per nome, tipologia, materia o data.
Gli eventi passati vengono nascosti automaticamente ma è possibile mostrarli su richiesta dell'utente.

Autenticazione

Nella schermata “Account” è stato integrato un semplice form di login che permette l’invio di credenziali da parte dell’utente per la registrazione o login al server Firebase, riconoscendo automaticamente se si tratta di un’utente già registrato o meno.
La particolarità di questo form consiste nel richiede il permesso all’utente di visualizzare i propri contatti, in modo da raccogliere le proprie email principali per mostrarle come scelte suggerite nella compilazione del campo riguardante l’indirizzo email.

Oltre all’autenticazione tramite email e password direttamente al server Firebase, è possibile effettuare l’accesso tramite un **provider**, ottenendo delle valide credenziali nel seguente modo:

- integrare il pulsante di reindirizzamento all’Activity auto-implementata, fornito dal provider, per esempio “Accedi con Facebook”;
- registrare la propria applicazione sul sito del provider, nel caso di Google questo passaggio viene automatizzato poichè l’applicazione è già stata registrata su Firebase, ma nel caso di Facebook è necessario registrare l’applicazione e fornire l’URL da cui verrà effettuata la richiesta di accesso alle informazioni personali dell’utente;
- recuperare l’**Identity Provider Token** ovvero un token con formato **OAuth 2.0** che verrà fornito dal provider una volta completata la procedura di accesso generata da quest’ultimo;
- utilizzare l’**Identity Provider Token** per richiedere l’accesso a Firebase, che verificherà la validità di questo e genererà un nuovo **Firebase ID Token**.

Una volta autenticati al server per la prima volta, l’applicazione provvederà automaticamente a ri-autenticare l’utente ad ogni avvio, occupandosi di conservare il **Firebase ID Token**.

Gestione dell’account

Sempre nella schermata “Account”, ma accessibile solo una volta effettuata l’autenticazione, è stato generata una schermata di impostazioni estendendo la classe FragmentPreference, contenente un layout auto-generato da Android per raccogliere graficamente e gestire le impostazioni definite nel file **account_preferences**, attraverso il quale l’utente può:

- modificare le informazioni riguardanti il proprio account;
- attivare e personalizzare funzionalità predefinite dell’applicazione;
- collegare il proprio account Spaggiari fornendo le dovute credenziali di accesso;

Interazione con ClasseViva Spaggiari

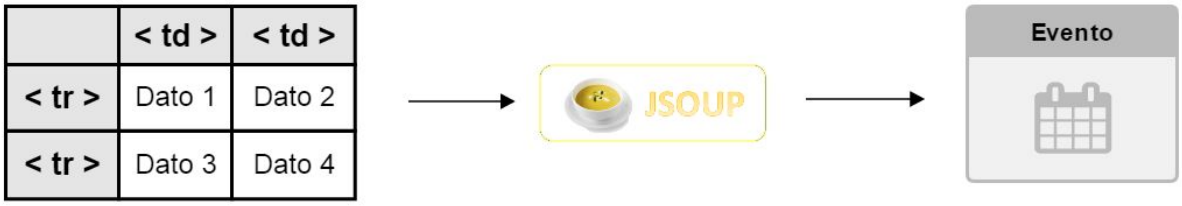
Nel caso l'utente colleghi il proprio account dell'applicazione ad un account Spaggiari, la classe SpaggiariManager, che estende Service di Android, si occuperà di gestire in background le richieste al server originale e di aggiornare periodicamente i dati locali dell'utente riguardanti i nuovi eventi creati nell'agenda di classe.

Per effettuare in modo semplice queste operazioni sono state utilizzate le librerie:

- **LoopJ** per effettuare richieste HTTP, che si occupa automaticamente della gestione dei cookie, utilizzando un contenitore di cookie persistente che risulta necessario per effettuare correttamente le successive richieste al server, mantenendo costante il cookie **PHPSESSIONID**.
- **JSoup** per effettuare lo **scraping** delle pagine web in Java e il **parsing** dei dati HTML.

Seguendo questo procedimento per richiede l'agenda di classe al Web Sever:

- effettuare la prima richiesta HTTP di autenticazione, la richiesta è di tipo GET e richiede come parametri il codice della scuola, il codice utente e la password dell'utente, in modo da ottenere l'ID di sessione;
- effettuare una richiesta per ottenere l'agenda di classe che verrà restituita sotto forma di tabella HTML;
- inizializzare la libreria JSoup con il body della risposta, ovvero la tabella contenuta nella pagina HTML, per effettuarne il parsing.
- iterare ogni cella della tabella istanziando nuovi oggetti di tipo **Event** tramite i dati ottenuti.



Scraping

Con Web Scraping (detto anche Web Harvesting o Web Data Extraction), si intende una tecnica informatica di estrapolazione dei dati da un sito web, messa in atto principalmente simulando la navigazione di un utente all'interno del World Wide Web.
In poche parole uno strumento di scraping si occupa di navigare in modo automatico all'interno di un sito web, estraendo le informazioni ritenute utili.
Eventualmente è possibile effettuare il **parsing** dei dati ricevuti, traducendoli in un formato più leggibile.

Parsing dei dati

Il parsing di dati consiste nell'analizzare un flusso di dati proveniente da una qualsiasi sorgente, per esempio un file, per determinarne la sua struttura rispetto ad una data grammatica e, generalmente, estrapolarne i dati necessari.

Gestione dello stato di autenticazione

Utilizzando un Service avviato all'apertura dell'applicazione che resta in ascolto sullo **stato di autenticazione** dell'utente al server Firebase, è possibile di notificare la classe User con il nuovo utente connesso in caso di login oppure, in caso di logout, effettuare automaticamente un login anonimo per garantire la sincronizzazione delle azioni effettuate anonimamenente, in caso di riconnessione tramite un altro account.

La classe statica User, aggiornata al cambiamento di stato dell'utente, contiene tutte le informazioni riguardanti quest'ultimo e gestisce la sincronizzazione dei dati tra il server Firebase e le strutture di memorizzazione offline.

Interazione con il database

In alternativa all'interazione diretta con il database tramite la formulazione di query specifiche, è stata utilizzata la libreria Java **ORMLite** che permette la creazione automatica di un database con tabelle **vincolate** alla struttura delle classi Java preimpostate per essere riconosciute dalla libreria.
Ogni classe le cui istanze vogliono essere memorizzate nel database come record di tabella, deve essere resa nota tramite **notazioni** Java specifiche, descrivendone il nome della tabella, il nome di ogni colonna corrispondente agli attributi della classe e, se necessario, ulteriori informazioni che specificano il tipo di valore corrispondente in SQL, il valore di default e altre utilità.

Una volta effettuata la configurazione, la libreria si occuperà di generare un file di testo contenente la struttura del database, con il quale è possibile interagire acquisendo un **Database Access Object** specifico per ogni tabella vincolata, ovvero specificandone la classe Java di appartenenza.
Questo permette in modo semplice di generare nuovi record semplicemente passando come parametro un'istanza della classe o di effettuare ogni tipologia di query utilizzando metodi java corrispondenti alle **key words** di SQL.

In particolare è stata creata una classe Singletone Java con diretto accesso ai metodi della libreria che si occupa dell'interazione con il database e della gestione di eventuali eccezioni.

Notifiche

Per la gestione delle notifiche di sistema è stato utilizzando lo strumento android NotificationManager che permette di schedulare l'invio di Intent Broadcast per la creazione di notifiche precisandone un orario.
Per la ricezione di questi eventi è stata estesa la componente Broadcast Receicever, configurandola in modo da intercettare Intent Broadcast con tipologia di azione corrispondente al filtro "**display notification**", per raccogliere dagli Extras le informazioni da visualizzare e creare una notifica di sistema.

Material Design

Nella creazione del layout dei Fragment dell'applicazione, delle animazioni e del comportamento dei componenti all'interazione dell'utente, sono state seguite le linee guida fornite da Google per il nuovo design **Material**, introdotto e supportato da versioni di Android 5.0 o superiori, le cui specifiche di implementazione sono pubbliche e consultabili online.

Implementazioni Future

Orario Scolastico

La prossima implementazione riguarderà la strutturazione di una schermata per la visualizzazione e la modifica dell'orario scolastico settimanale tramite il drag&drop delle materie precedentemente inserite.

Supporto per versioni precedenti di Android

Attualmente l'applicazione è supportata **logicamente** e **graficamente** da versioni di Android superiori alla 5.0 Lollipop a causa dell'utilizzo di componenti grafici orientati al Material Design.
Android mette a disposizione metodi per il controllo della versione del sistema operativo corrente, rendendo facile l'utilizzo di componenti diversi adatti al supporto di sistemi più datati, perciò l'applicazione verrà ampliata in modo da supportare versioni di Android 4.4 Kitkat o superiori, aprendosi al 77% dei dispositivi con a bordo Android.

Pubblicazione

L'applicazione si manifesta funzionante nel suo complesso e adatta all'utilizzo di un numero contenuto di utenti, ma allo stesso tempo ancora molto immatura e bisognosa di ottimizzazioni per quanto riguarda le funzionalità e le prestazioni generali.
Una volta effettuate le dovute implementazioni in grado di supportare una ben più ampia gamma di utenti, l'applicazione verrà pubblicata sul **Google Play Store**.

Sitografia

Strumenti

Android Studio: https://developer.android.com/studio/index.html
Ambiente di sviluppo ufficiale per la piattaforma Android.
Material Design: https://material.google.com/
Linee guide per la realizzazione di un'applicazione con design Material.
Firebase: https://firebase.google.com/
Server web che garantisce numerose funzionalità per lo sviluppo.
Genymotion: https://www.genymotion.com/
Software di emulazione di dispositivi Android.
Git: https://git-scm.com/
Sofware di versioning del codice.

Librerie

GSON: https://github.com/google/gson
Serializzazione e deserializzazione di classi Java in formato JSON e viceversa.
Material Calendar View: https://github.com/prolificinteractive/material-calendarview
Calendario mensile scorrevole personalizzabile.
Circle Image View: https://github.com/hdodenhof/CircleImageView
Contenitore di immagini con adattamento circolare.
ORMLite: http://ormlite.com/
Utilità di interazione con SQLite e vincolazione delle tabelle SQL alle classi Java.
FirebaseUI: https://github.com/firebase/FirebaseUI-Android
Componenti grafici classici adattati all'interazione realtime con il database di Firebase.
Facebook for Developers SDK: https://developers.facebook.com/
SDK fornite da Facebook per l'utilizzo di funzionalità di interazione con esso.
LoopJ: http://loopj.com/android-async-http/
Utilità per richieste HTTP asincrone.
JSoup: https://jsoup.org/
Utilità per lo scraping di siti web.