1 Descrizione delle classi principali

Di seguito vengono riportate le descrizioni delle classi più importanti dell'applicazione.

1.1 dependency_injection::AppModule

La classe `AppModule` si occupa della risoluzione delle dipendenze per permettere l'utilizzo della dependency injection. La libreria Dagger 2 utilizza i metodi di `AppModule` con l'annotazione *@Provides* per l'inizializzazione dei campi dati annotati con *@Inject* delle varie classi dell'applicazione. Questo meccanismo è utilizzato per far corrispondere per ogni campo dati che ha come tipo un interfaccia, l'istanza di una classe che implementa tale interfaccia.

1.2 model::dao::Content

La classe `Content` è utilizzata per la gestione dei contenuti xAPI, di cui l'applicazione deve permettere la fruizione. Gli oggetti di tale classe rappresentano una entry nella relazione *Content* nel database locale. Per questo motivo gli attributi della relazione *Content* e i campi dati della relazione omonima corrispondono. La classe `Content` estende la classe `RealmObject`, fornita dalla libreria Realm. Grazie a ciò è possibile modificare i campi dati di un oggetto di tipo `Content`, ripercuotendo tali modifiche sulla entry corrispondente del database locale. In modo analogo anche le classi `ContentVersion`, `LrsData`, `History` e `UserData` estendono `RealmObject`, permettendo quindi le modifiche ai valori della relazione con nome corrispondente.

1.3 model::data_access::DatabaseAccess

La classe `DataAccess` è utilizzata per la gestione dei dati del database riguardanti i contenuti e i dati di fruizione ad essi associati. Tale classe implementa le interfacce `ContentDataAccess` e `ContentHistoryForLoggedUserAccess` le quali espongono, rispettivamente, i metodi per l'inserimento, recupero e modifica dei contenuti xAPI trattati dall'applicazione e dei dati di fruizione di un determinato contenuto da parte dell'utente loggato. Tale classe in particolare si occupa anche di inserire nel database del dispositivo i dati dei contenuti da visualizzare nell'applicazione.

1.4 model::data_access::StatementSenderImp

La classe `StatementSenderImp` si occupa della gestione e dell'invio degli statement ottenuti dall'interazione di un utente con i contenuti xAPI al LRS. Tale classe implementa l'interfaccia `StatementSender`, che a sua volta estende le interfacce `DatabaseChangeListener` e `NetworkCHangeListener`. L'interfaccia `StatementSender` espone i metodi che devono essere implementati al fine di permettere l'invio degli statement ad un LRS. Le interfacce `DatabaseChangeListener` e `NetworkCHangeListener` permettono, invece, agli oggetti di tale classe di registrarsi come "listener" dei cambiamenti nel database degli statement, gestito dalla libreria TinCanAndroid-Offline, e nella connettività ad internet del dispositivo. Nel caso in cui sia presente una connessione internet e vi siano statement non inviati al LRS tale classe provvederà ad inviarli, eliminandoli dal database della libreria TinCanAndroid-Offline. Per fare questo gli oggetti a tale classe hanno un riferimento ad un oggetto di tipo `RSTinCanOfflineConnector`. Tale oggetto, della libreria TinCanAndroid-Offline, permette sia l'accesso al database degli statement, sia l'invio di quest'ultimi al LRS.

1.5 model::lrs_access::LrsSynchronizeImp

La classe `LrsSynchronizeImp` si occupa di effettuare delle richieste al LRS per la il recupero di dati di fruizione di un utente che non sono presenti localmente. Gli oggetti di questa classe recuperano gli URL per la comunicazione con LRS utilizzando un oggetto sottotipo dell'interfaccia `LrsRequestUrl`. Nel caso sia disponibile una connessione ad internet, le istanze di questa classe si occupano di effettuare le richieste per recuperare le informazioni desiderate dal LRS. Le informazioni ricavate dalla risposta del LRS sono utilizzate per aggiornare i dati di fruizione di un determinato utente, presenti nel database locale, utilizzando un oggetto di tipo `ContentHistoryForLoggedUserAccess`. Utilizzando come LRS Learning Locker, le risposte ottenute sono in formato JSON e vengono gestite utilizzando la libreria Gson.

1.6 model::result::StartResult

La classe `StartResult` è utilizzata per ricavare le informazioni che vengono inviate dallo script JavaScript, che si occupa di ricavare i dati di fruizione di un utente, riguardo all'inizio di un certo contenuto, da parte di un utente. Il codice di tale classe è generato utilizzando la libreria Google AutoValue. Praticamente questa libreria si occupa di implementare i metodi delle classi che hanno l'annotazione *@AutoValue*, fornendo anche la possibilità di creare

una classe interna "Builder", per l'implementazione del design pattern Builder appunto. Il programmatore non deve far altro che dichiarare la classe che si vuole creare e la una classe interna `Builder` come astratte, dichiarare tutti i metodi getter per i campi dati, anch'essi astratti, e dichiarare i metodi della classe `Builder` per settare i campi dati, sempre astratti. Le istanze di questa classe vengono create sfruttando un oggetto `JsonObject`, il quale contiene le informazioni ricavate dallo script JavaScript. Anche le classi `PartialResult` e `TotalResult` sono state create utilizzando la libreria Google AutoValue e hanno scopi simili alla classe `StartResult`.

1.7 model::statement_object::StartStatementImp

La classe `StartStatementImp` rappresenta uno statement di inizio di un contenuto da parte di un utente, che deve essere inviato ad un LRS. Tale classe implementa l'interfaccia `StartStatement` e estende `Statement` della libreria TinCanAndroid-Offline. I campi dello statement relativi a chi ha effettuato l'azione vengono riempiti utilizzando un oggetto di tipo `UserDataAccess`, il quale permette di accedere ai dati dell'utente loggato. Il campo *verb* viene inizializzato utilizzando la definizione del verbo *inizialized*, disponibile all'URL `http://adlnet.gov/expapi/verbs/inizialized`. Infine i campi che definiscono il contesto dello statement e il campo *object* vengono riempiti utilizzando i campi dati dell'oggetto `StartResult`, richiesto dal costruttore della classe. Le classi `PartialResulStatementImp` e `TotalResulStatementImp` hanno compiti analoghi e vengono sono costruite in modo simile.

1.8 presenter::javascript_communication:: MyJavascriptInterfaceImp

La classe `MyJavascriptInterfaceImp` si occupa di ricevere i dati di fruizione di un utente da parte dello script Javascript. Tale classe, inoltre, si occupa di trasformare in statement questi dati e di inserirli nel database degli statement, utilizzando un oggetto `RSTinCanOfflineConnector`, classe della libreria TinCanAndroid-Offline. Ad ogni inserimento, inoltre, viene inviato un messaggio in broadcast utilizzando un `LocalBroadcastManager`, classe del framework Android, in modo tale che le classi che sono in ascolto per i cambiamenti del database degli statement siano avvertite. Per fare ciò implementa l'interfaccia `MyJavascriptInterface`, i cui metodi che possono essere richiamati da JavaScript sono annotati con *@JavascriptInterface*. Tale annotazione permette l'invocazione di metodi delle classi di Android da JavaScript, anche per le versioni uguali o inferiori ad Android 4.2(API level 18).

Tale classe, infine, ha il compito di aggiornare le informazioni di fruizione dei contenuti xAPI dell'utente loggato, in base ai dati ricevuti.

1.9 presenter::manager::DownloaderManagerImp

La classe `DownloaderManagerImp` si occupa del download delle informazioni relative ai contenuti da mostrare nell'applicazione e di renderli disponibili anche in assenza di connessione internet. Per permettere la fruizione di un determinato contenuto in modalità offline, le istanza di tale classe si connettono al server in cui il contenuto risiede per effettuare il download di alcuni file compressi. Tali file specificano le risorse necessarie per la riproduzione del contenuto e di ogni singola slide. Una volta scaricati, vengono decompressi e viene estratto dal loro interno un file in formato XML. Quest'ultimi contengono dei riferimenti a tutte le risorse di cui fare il download per permettere la riproduzione di un contenuto. Le informazioni presenti nei file XML vengono estratte utilizzando delle espressioni XPath.

1.10 presenter::receiver::InternetStateReceiverImp

La classe `InternetStateReceiver` si occupa di avvisare ad ogni variazione nella connettività ad internet del dispositivo tutti gli oggetti che implementano l'interfaccia `NetworkChangeListener`, che si sono registrati come "listener". Tale classe implementa l'interfaccia `InternetStateReceiver`, che espone i metodi per registrarsi e deregistrarsi come "listener", ed estende la classe astratta `BroadcastReceiver`, del framework Android, della quale implementa il metodo *onReceive*, invocato ad ogni variazione nello stato della connessione ad Internet. Il funzionamento e lo scopo della classe `DatabaseStateReceiverImp` sono analoghi.

1.11 presenter::MyApplication

La classe `MyApplication` estende la classe `Application` del framework Android e si occupa di fornire alcuni metodi statici di utilità. Questi metodi permettono di accedere al contesto di esecuzione, allo stato della connessione internet e alla componente necessaria per effettuare la dependency injection.

1.12 presenter::HomeActivityPresenterImp

La classe `HomeActivityPresenterImp` si occupa, interagendo con le componenti dei package `model` e `presenter`, del recupero delle informazioni che devono essere visualizzate dalla classe `HomaActivity` e della ge-

stione delle richieste di tale classe, provenienti dall'interazione dell'utente con l'interfaccia grafica. In particolare si occupa di fornire i dati relativi a tutti i contenuti xAPI che devono essere riprodotti dall'applicazione, di gestire le richieste di download e rimozione di tali contenuti e della gestione del logout di un utente. `HomeActivityPresenterImp` implementa l'interfaccia `HomeActivityPresenter`. In modo analogo le classi `ContentSelectedPresenterImp`, `ImageAdapterPresenterImp` e `ReportActivityPresenterImp` si occupano, rispettivamente, della gestione e del recupero delle informazioni che devono essere visualizzate per le classi `ContentAlertDialog`, `ImageAdpater` e `ReportActivity`.

1.13 view::HomeActivity

La classe `HomeActivity` si occupa della gestione dell'interfaccia grafica dell'home page dell'applicazione. Tale classe ha il compito di mostrare i corretti oggetti grafici e di reagire alle interazioni dell'utente con l'interfaccia, richiamando i metodi di un'istanza della classe `HomeActivityPresenterImp` per la gestione della richiesta. In particolare tale classe mostra, con una disposizione a griglia, i corsi in formato xAPI che possono essere riprodotti utilizzando l'applicazione. Sui contenuti è possibile effettuare un tap, per accedere alla schermata che permette di avviare la riproduzione del contenuto e il download, oppure tenendo premuto su di essi è possibile selezionarli, per effettuare il download o la cancellazione di uno o più contenuti. Inoltre si occupa di visualizzare un menu per accedere all'area di report, ripristinare i contenuti cancellati e effettuare il logout.