Università degli Studi di Padova

Dipartimento di matematica

Corso di Laurea in Informatica

Titolo della tesi

Relatore

Prof.ssa Ombretta Gaggi

Laureando

Marco Zanella
1074420

Anno Accademico 2015/2016

Sommario

Il presente documento ha lo scopo di descrivere l'attività di stage svolta presso l'azienda *Modo Network s.r.l.* con sede a Caerano San Marco(TV) tra maggio e luglio 2016.
Il lavoro si è concentrato sulla progettazione e l'implementazione di un'applicazione Android per la fruizione e il tracciamento di contenti per l'e-learning in formato xAPI.

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo dello stage

L'**Experience API**(xAPI anche conosciuta come Tin Can API) è una specifica software per la creazione di contenuti per l'e-learning. È considerata il successore dello **SCORM**, ovvero lo standard de facto utilizzato per l'inserimento dei contenuti di e-learning all'interno di **Learning Management Systems**. Tali piattaforme sono necessarie per la distribuzione dei contenuti in formato SCORM e sono normalmente accessibili utilizzando un browser, per la visualizzazione dei contenuti in esse pubblicati. Con la nuova specifica, però, non sono più necessarie: i contenuti xAPI sono fruibili anche senza l'utilizzo di browser e di connessioni internet. Le informazioni sulle esperienze degli utenti sono tracciate utilizzando degli statement in formato JavaScript Object Notation(JSON) e sono raccolti da un **Learning Record Store**(LRS). Gli statement sono trasmessi agli LRS utilizzando connessioni HTTP o HTTPS, che provvedono a salvarli. Nella loro forma più semplice si compongono di tre parti:

- *attore*: specifica chi ha compiuto una determinata azione;
- *verbo*: l'azione compiuta dall'attore;
- *oggetto*: l'oggetto a cui è rivolta l'azione.

Sfruttando tale specifica si vuole creare un'applicazione Android che permetta la fruizione di contenuti per l'e-learning sia quando il dispositivo è online che offline. Quando il dispositivo non ha una connessione internet attiva le esperienze di un utente dovranno essere salvate sul dispositivo in modo tale che, quando sarà nuovamente disponibile una connessione, sia possibile inviare tali statement ad un LRS. L'applicazione deve inoltre presentare la possibilità di specificare quali contenuti devono essere visualizzati.

1.2 Struttura del documento

In questa parte viene riportata la struttura del documento. Ad ogni sezione è associata una breve descrizione del contenuto.

1. Introduzione: scopo dello stage e contenuti presenti nel documento;
2. Progetto dello stage: descrizione degli obiettivi dello stage e delle caratteristiche che il prodotto finale deve avere;

3. Tecnologie utilizzate: descrizione delle tecnologie utilizzate durante lo stage;
4. Progettazione: descrizione delle scelte progettuali fatte riguardanti l'applicazione;
5. Sviluppo: descrizione della fase di sviluppo dell'applicazione;
6. Verifica e validazione: descrizione della fase di testing dell'applicazione.

2 Progetto dello stage

2.1 Motivazione

La specifica xAPI ha aperto una nuova serie di possibilità per la fruizione di contenuti per l'e-learnig. Infatti non essendo più legati alla necessità di avere una connessione internet continua è possibile creare delle applicazioni per dispositivi mobile, che permettano il download di tali contenuti, cosicché un utente possa fruirne in qualsiasi momento. Inoltre le persone si stanno abituando sempre più all'utilizzo delle app, le quali riscontrano successo poichè riducono il tempo di accesso alle informazioni desiderate. Nel caso specifico un'applicazione può evitare ad un utente di dover aprire un browser, cercare su di un motore di ricerca la piattaforma LMS a cui collegarsi oppure scriverne l'URL, entrare nella piattaforma e, eventualmente, effettuare il login. Sfruttare la specifica xAPI inoltre può risultare conveniente anche per chi richiede contenuti per l'e-learning: infatti le piattaforme LMS solitamente sono a pagamento e grazie a xAPI non sono più strettamente necessarie ed è possibile sfruttare un qualsiasi spazio web per la pubblicazione dei contenuti, mentre per recuperare le esperienze degli utenti si utilizza un LRS, che non ha necessità particolari e ne sono disponibili diversi che sono open source (per esempio Learning Locker). Infine è utile anche a chi eroga corsi di e-learning, poichè è possibile fornire ai clienti un'ulteriore strumento per la fruizione dei contenuti.

2.2 Descrizione del prodotto finale

Nome Applicazione consiste appunto in un'applicazione per la fruizione e il tracciamento di contenuti xAPI sia online che offline. I contenuti che possono essere fruiti dall'applicazione sono specificati in un file JSON che risiede in un server raggiungibile online. La locazione di tale file deve essere specificata in un file di configurazione dell'applicazione stessa. L'applicazione inoltre permette l'autenticazione e la registrazione: ciò è fondamentale per poter associare ogni azione significativa che dev'essere tracciata alla persona che l'ha eseguita. Ad ogni accesso, in caso di connessione internet attiva, l'applicazione controlla:

- se sono disponibili aggiornamenti ai contenuti disponibili: in tal caso viene aggiornato il database locale che tiene conto dei contenuti disponibili;
- se sono state registrate nel LRS azioni di un certo utente di cui non è presente traccia in locale: in tal caso vengono aggiornati i dati relativi

alle esperienze di un utente, anch'essi registrati nel database presente nel dispositivo.

Nome Applicazione permette ad un utente autenticato di accedere ad un riepilogo delle attività riguardanti i contenuti di e-learning presentati nell'applicazione.

2.3 Obiettivi

Gli obiettivi dello stage sono stati individuati assieme al tutor aziendale Germana Boghetto e al responsabile dell'area informatica Marco Petrin. Sono stati suddivisi in:

- obiettivi obbligatori: requisiti minimi che devono essere soddisfatti dall'applicazione alla fine dello stage;
- obiettivi opzionali.

2.3.1 Obiettivi obbligatori

Identificativo	Descrizione
OB.1	L'applicazione deve permettere la visualizzazione di oggetti didattici in formato xAPI
OB.1.1	L'applicazione deve permettere la fruizione degli oggetti didattici sia in modalità online che offline del dispositivo
OB.2	L'applicazione deve consentire l'interazione tra l'utente e l'oggetto didattico come da funzionalità implementate nell'oggetto didattico stesso
OB.3	L'applicazione deve tracciare i dati di fruizione dell'utente all'interno dell'oggetto didattico
OB.3.1	L'applicazione deve permettere di registrare i dati di più oggetti didattici differenti
OB.3.2	L'applicazione deve permettere di estrarre, inviare e/o visualizzare i report di fruizione degli utenti sui vari oggetti didattici
OB.4	L'applicazione deve funzionare su dispositivi Android

2.3.2 Obiettivi opzionali

Identificativo	Descrizione
OP.1	L'applicazione può essere personalizzabile graficamente a seconda delle esigenze del cliente
OP.1.1	La personalizzazione grafica può comprendere la modifica dei colori, di un eventuale logo e dei font presenti all'interno dell'applicazione
OP.2	L'applicazione può permettere la profilazione di diversi utenti
OP.2.1	La profilazione può essere determinata da una piattaforma che sta a monte dell'oggetto didattico
OP.2.2	La profilazione con utenti diversi comporta diversi report all'interno dell'applicazione

2.4 Vincoli tecnologici

Non è stato imposto alcun vincolo particolare sulle tecnologie da utilizzare nello sviluppo dell'applicazione. Le uniche tecnologie di cui è richiesto l'utilizzo sono:

- Android;
- contenuti xAPI.

3 Tecnologie utilizzate

3.1 Android

Android è un sistema operativo mobile sviluppato da Google e basato su kernel Linux. È stato progettato per essere eseguito principalmente su smartphone e tablet, con interfacce utente specializzate per orologi e televisori. Le versioni supportate dall'applicazione sono la 4.2 e superiori.

3.1.1 Vantaggi

- possiede una vasta fetta di mercato mobile;
- disponibile su un vasto numero di dispositivi;
- quasi totalmente gratuito ed open-source.

3.1.2 Svantaggi

- essendoci un vasto numero di produttori di smartphone e tablet che non aggiornano la versione di Android che rilasciano all'interno dei loro dispositivi, Android risulta essere estremamente frammentato.

3.2 Dagger 2

Dagger 2 è una libreria che permette la dependency injection in Android. Nel progetto è largamente utilizzato tale design pattern per diminuire il più possibile le dipendenze e ciò permette all'applicazione di essere facilmente sviluppata dopo lo stage. La scelta di tale libreria è stata fatta poichè permette di effettuare l'injection di una qualsiasi classe, a differenza di altre di librerie come RoboGuice o Butterknife.

3.2.1 Vantaggi

- non rallenta l'applicazione a runtime;
- le risoluzioni delle dipendenze possono essere dichiarate come dei comuni metodi, sfruttando le appropriate annotazioni.

3.2.2 Svantaggi

- non è sempre riconosciuto subito dagli IDE, che possono scambiare l'utilizzo di tale libreria per errori;

- i campi che devono essere inizializzati tramite dependency injection non possono avere visibilità privata o protetta.

3.3 Google AutoValue

Google AutoValue è una libreria che permetta la generazione automatica del codice delle classi annotate tramite delle annotazioni proprie della libreria. Tale libreria permette di creare gli oggetti di tale classi utilizzando il design pattern Builder. È utilizzata per le classi che potrebbero richiedere molti parametri nel costruttore, evitando così il fenomeno del telescoping.

3.3.1 Vantaggi

- evita al programmatore la creazione di classi semplici ma ripetitive, diminuendo la possibilità di errori;
- mette a disposizione il pattern Builder per la creazione di oggetti;
- le classi vengono generate in fase di compilazione.

3.3.2 Svantaggi

- le classi create non sono disponibili tra quelle visibili nel sorgente;
- non è sempre riconosciuto subito dagli IDE, che possono scambiare l'utilizzo di tale libreria per errori.

3.4 Gson

Gson è una libreria per la gestione di oggetti in formato JSON in Android. Tale libreria è utilizzata sia per la lettura dei contenuti disponibili pubblicati sul server, sia per la trasformazione degli oggetti scambiati tra Java e JavaScript. È stata scelta tale libreria per la semplicità si utilizzo.

3.4.1 Vantaggi

- molto intuitiva;
- permette la trasformazione di oggetti Java in oggetti in formato JSON e viceversa.

3.4.2 Svantaggi

- più lenta di altre librerie che offrono le stesse funzionalità.

3.5 Java

Java è uno dei più famosi linguaggi di programmazione orientato agli oggetti supportato da una moltitudine di librerie e documentazione. Viene utilizzato per la scrittura e lo sviluppo dell'applicazione.

3.5.1 Vantaggi

- linguaggio più conosciuto, diffuso e utilizzato nell'ambiente di sviluppo Android
- ampia documentazione disponibile;
- dispone di un gran numero di librerie;
- portabilità su diversi sistemi operativi.

3.5.2 Svantaggi

- linguaggio verboso.

3.6 JavaScript

JavaScript è un un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso. L'utilizzo di tale tecnologia si è rilevata necessaria per recuperare le informazioni di fruizione dei contenuti di un utente e mandarle all'applicazione Android.

3.6.1 Vantaggi

- permette l'interazione con pagine web e con i contenuti xAPI;
- è eseguito lato client.

3.6.2 Svantaggi

- problemi di sicurezza, soprattutto in ambiente Android.

3.7 MPandroidChart

MPAndroidChart è una libreria per la creazione di grafici in Android. Essa viene usata per la creazione di grafici a torta nell'applicazione, ma ne supporta molti altri. Tale libreria fornisce anche delle funzioni di ridimensionamento, trascinamento, selezione e di animazione. Tale scelta è stata fatta principalmente per le poche alternative disponibili.

3.7.1 Vantaggi

- creazione semplice di grafici;
- grafici ampiamente personalizzabili;
- legende create automaticamente e personalizzabili.

3.7.2 Svantaggi

Non sono stati riscontrati particolari svantaggi nell'utilizzo di tale libreria, poichè accompagnata da una buona documentazione.

3.8 Picasso

Picasso è una libreria Android utile per download, caching e mostrare le immagini. Tale libreria è stata utilizzata per la gestione di tutte le immagini relative ai contenuti presenti nell'applicazione.

3.8.1 Vantaggi

- utilizzo molto semplice;
- permette di gestire immagini nella memoria del dispositivo o online indifferentemente.

3.8.2 Svantaggi

- qualche problema nel ridimensionamento delle immagini per inserirle nelle ImageView di Android.

3.9 Realm

Realm è una libreria che offre le funzionalità di utilizzo di un database con le operazioni CRUD e la trasformazione di oggetti del database in oggetti Java in modo automatico.

3.9.1 Vantaggi

- semplicità di utilizzo;
- accesso veloce ai dati;
- thread-safe;
- trasformazione automatica tra oggetti del database e oggetti Java.

3.9.2 Svantaggi

- non è intuitivo il comportamento nel multithreading;
- tutti gli oggetti che devono essere salvati nel database devono estendere una specifica classe.

3.10 TinCanAndroid-Offline

TinCanAndroid-Offline è una libreria utilizzata per la creazione, il salvataggio in locale e l'invio di statement in formato xAPI. È stata adottata poichè è l'unica libreria open-source che fornisce tale funzionalità e non sono state trovate alternative valide nemmeno tra le librerie non disponibili gratuitamente.

3.10.1 Vantaggi

- open-source;
- non sono state trovate delle valide alternative;
- è possibile accedere al codice sorgente su Github;
- gestisce in maniera abbastanza semplice il salvataggio in locale e l'invio degli statement.

3.10.2 Svantaggi

- documentazione praticamente assente;
- pochi esempi di utilizzo disponibili;
- non è disponibile nei repository dove vengono pubblicate le librerie di cui è possibile semplicemente dichiarare la dipendenza nel file Gradle per la compilazione di un progetto Android Studio;

- non è disponibile un file JAR da includere in un progetto Android;
- non sempre intuitiva nell'utilizzo.

4 Progettazione

4.1 Architettura ad alto livello

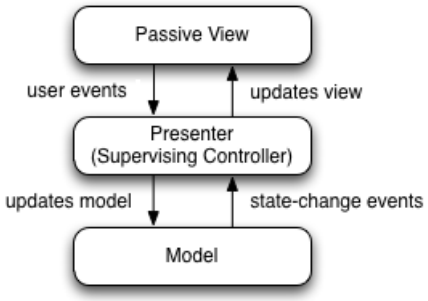

Figura 1: Rappresentazione pattern MVP

L'architettura dell'applicazione segue il pattern architetturale Model-View-Presenter, utilizzato insieme alla dependency injection. Tale scelte, unite ad un uso diffuso di interfacce permettono di ridurre il grado di accoppiamento tra le parti che l'applicazione in modo tale da aumentarne il più possibile l'estensibilità.

Model Il model rappresenta i dati che vengono trattati all'interno dell'applicazione. Le classe appartenenti al model rappresentano:

- i dati che vengono salvati in modo persistente nel database presente nel dispositivo in cui è installata l'applicazione;
- gli statement che devono essere inviati ad un LRS;
- gli oggetti che vengono scambiati tra JavaScript e Android per il tracciamento delle esperienze di un utente;
- oggetti che permettono l'accesso a tali dati.

Nell'applicativo è rappresentato dal package omonimo.

Presenter Il presenter si occupa di recuperare i dati del model e passarli alla view per essere mostrati. Esso si occupa inoltre di gestire le richieste della view in seguito ad una interazione con un utente.
Nell'applicativo è rappresentato dal package omonimo.

View La view si occupa della dell'interfaccia grafica dell'applicazione e di notificare al presenter le interazioni dell'utente, che ha il compito di elaborare. Nell'applicativo è rappresentato dal package omonimo.

4.2 Persistenza dei dati

4.2.1 Il database locale

L'applicazione si appoggia su di un database creato utilizzando la libreria Realm per il salvataggio dei dati relativi agli utenti, ai contenuti e alla loro fruizione in un database locale. In tale database vengono salvati i dati relativi ai contenuti xAPI che devono essere visualizzati la prima volta che l'applicazione viene avviata, mentre i dati relativi all'utente vengono salvati quando viene effettuato un login alla registrazione. I dati di fruizione, invece, all'interazione di un utente con un contenuto.

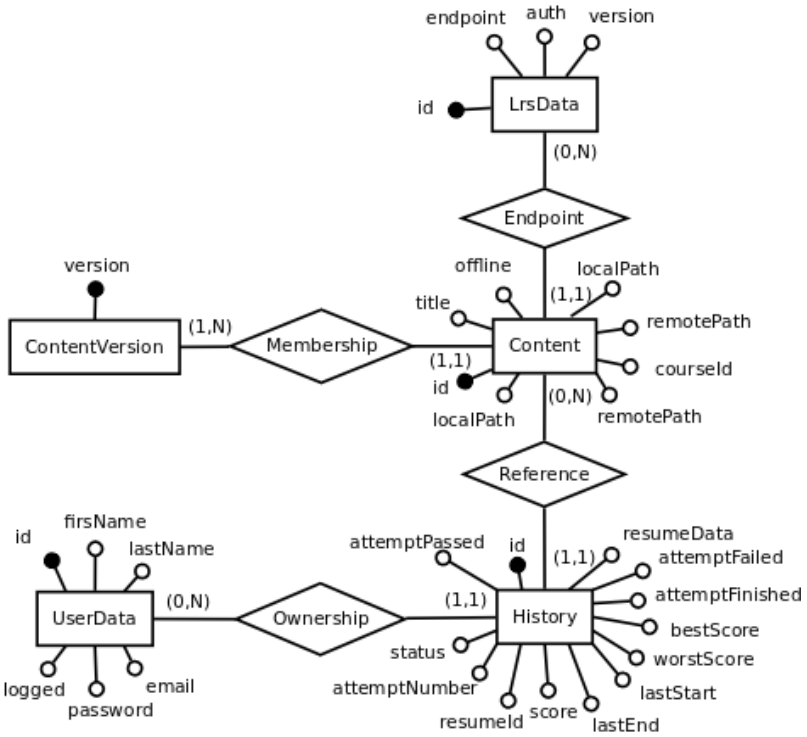

Figura 2: Diagramma ER del database locale

4.2.1.1 Diagramma ER

4.2.1.2 Descrizione delle relazioni

Content Relazione che contiene le informazioni dei contenuti xAPI che devono essere gestiti dall'applicazione. Attributi:

- **id**: chiave primaria, intero;
- **title**: stringa, rappresenta un titolo associato al contenuto;
- **remotePath**: stringa, rappresenta l'URL a cui è possibile recuperare il contenuto;
- **localPath**: stringa, rappresenta il percorso locale al dispositivo a cui è possibile recuperare il contenuto, se questo è disponibile offline;
- **offline**: boolean, indica se il contenuto è disponibile per la riproduzione offine oppure no;
- **courseId**: stringa, rappresenta l'identificativo utilizzato dal LRS per distinguere un contenuto;
- **lrsDataId**: intero, chiave esterna verso la relazione LrsData.

ContentVersion Relazione che serve per specificare la versione dei contenuti trattati dall'appliaczione. Attributi:

- **version**: chiave primaria, stringa, rappresenta la versione dei contenuti.

LrsData Relazione che contiene le informazioni riguardanti un LRS a cui inviare i dati di fruizione dei contenuti di un certo utente. Attributi:

- **id**: chiave primaria, intero;
- **endpoint**: stringa, rappresenta l'URL a cui inviare gli statement in formato xAPI;
- **auth**: stringa, rappresenta metodo e dati per l'accesso al LRS;
- **version**: stringa, rappresenta la versione di statement xAPI accettata dal LRS.

History Relazione che contiene i dati di fruizione di ogni contenuto per gli utenti che hanno utilizzato l'applicazione su di un determinato dispositivo. Attributi:

- **id**: chiave primaria, intero;
- **resumeId**: stringa, identificativo utilizzato dalla componente che si occupa della riproduzione dei contenuti xAPI per l'accesso ai dati per riprendere un contenuto da dove lo si aveva interrotto all'ultima fruizione;
- **resumeData**: stringa, dati per riprendere un contenuto da dove lo si aveva interrotto all'ultima fruizione;
- **status**: intero, reppresenta lo stato in cui si trova un contenuto per un certo utente. Può rappresentare che un utente non ha mai acceduto ad un certo contenuto, che vi ha acceduto ma il contenuto non è stato fruito fino alla fine oppure che un utente ha superato o meno un certo contenuto;
- **score**: double, rappresenta l'ultimo punteggio ottenuto da un utente per un certo contenuto;
- **bestScore**: double, rappresenta il miglior punteggio ottenuto da un utente per un certo contenuto;
- **worstScore**: double, rappresenta il peggior punteggio ottenuto da un utente per un certo contenuto;
- **lastStart**: long, timestamp in millisecondi dell'ultima volta che un utente ha iniziato un certo contenuto;
- **lastStart**: long, timestamp in millisecondi dell'ultima volta che un utente ha terminato un certo contenuto;
- **attemptNumber**: intero, rappresenta il numero di volte che un utente ha iniziato un certo contenuto;
- **attemptFinished**: intero, rappresenta il numero di volte che un utente ha terminato un certo contenuto;
- **attemptPassed**: intero, rappresenta il numero di volte che un utente ha superato un certo contenuto;

- **attemptFailed**: intero, rappresenta il numero di volte che un utente non ha superato un certo contenuto;
- **contentId**: intero, chiave esterna verso la relazione Content. Rappresenta il contenuto a cui fanno riferimento i dati;
- **userId**: intero, chiave esterna verso la relazione UserData. Rappresenta l'utente a cui fanno riferimento i dati.

UserData Relazione che contiene le informazioni degli utenti che hanno utilizzato l'applicazione su di un determinato dispositivo. Attributi:

- **id**: chiave primaria, intero;
- **logged**: boolean, indica se un utente è loggato in quel dispositivo o meno;
- **email**: stringa, rappresenta l'email di un utente;
- **password**: stringa, rappresenta la password di un utente;
- **firstName**: stringa, rappresenta il nome di un utente;
- **lastName**: stringa, rappresenta il cognome di un utente.

4.2.1.3 Descrizione delle associazioni

Membership Associazione che unisce ogni contenuto alla sua versione.
Molteplicità:(1,N) ogni versione può essere associato a più contenuti, ogni contenuto può avere un'unica versione.

Endpoint Associazione che unisce ogni contenuto al LRS a cui devono essere inviati i dati di fruizione.
Molteplicità:(1,N) ogni LrsData può essere associato a più contenuti, ogni contenuto si riferisce ad un unico LrsData.

Reference Associazione che unisce ogni contenuto ai dati di fruizione per un certo utente.
Molteplicità:(1,N) ogni contenuto può essere associato a più History, ogni History può essere associata ad un solo contenuto.

Ownership Associazione che unisce ogni utente ai dati di fruizione di un certo contenuto.
Molteplicità:(1,N) ogni utente può essere associato a più History, ogni History può appartenere ad un unico utente.

4.2.2 Persistenza degli statement

Gli statement creati dall'applicazione per registrare i dati di fruizione dei contenuti xAPI sono salvati temporaneamente all'interno di un database SQLite, completamente gestito dalla libreria TinCanAndroid-Offline. Quando l'applicazione registra che è disponibile una connessione internet si occupa di inviare tali statement ad un LRS. Anche l'invio è gestito dalla libreria TinCanAndroid-Offline, che provvede, nel caso in cui non vi siano errori, ad eliminare dal database locale tutti gli statement inviati. Ad ogni avvio l'applicativo ha il compito di interrogare LRS a cui sono stati inviati i dati di fruizione al fine di verificare se vi siano dati che non sono presenti in locale: se presenti il database locale viene aggiornato. Poichè LRS utilizzato è Learning Locker, il quale utilizza come DBMS MongoDB, le richieste per recuperare dati dal LRS saranno delle query MongoDB e i risultati di tali richieste saranno in formato JSON.

4.2.3 Autenticazione e registrazione

L'autenticazione e la registrazione all'applicazione avvengono mediante delle richieste HTTP a due URL specificati nel file di configurazione dell'applicativo. Tale scelta è stata fatta per permettere di cambiare agevolmente tali indirizzi senza dover modificare il codice.
In fase di sviluppo sono stati creati anche due script PHP ed un database per gli utenti dell'applicazione. Tali script si occupano rispettivamente del login e della registrazione, interrogando il database degli utenti e restituendo il risultato della richiesta.

5 Descrizione dei package

Di seguito vengono presentati i package che compongono l'applicazione. Le componenti riportati in colori differenti dal giallo sono evidenziate poiché appartenenti a librerie esterne.

5.1 Package model

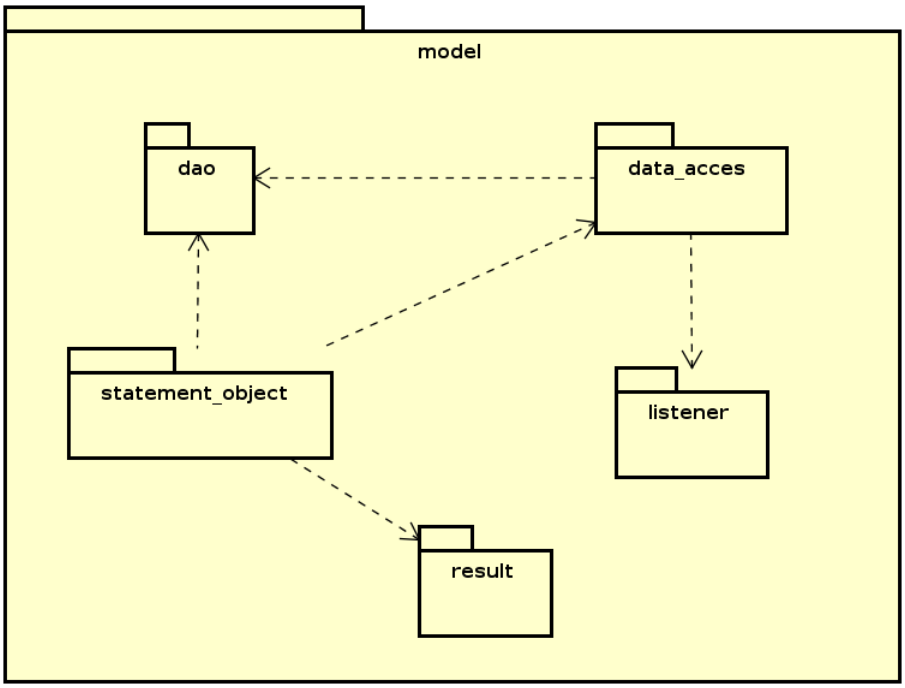

Figura 3: Package model

Il package `model` racchiude tutte le componenti che si occupano della rappresentazione dei dati trattati dall'applicazione, della loro memorizzazione e recupero.
I package contenuti al suo interno sono:

- `dao`;
- `data_access`;
- `statement_object`;
- `listener`;
- `result`.

5.2 Package model::dao

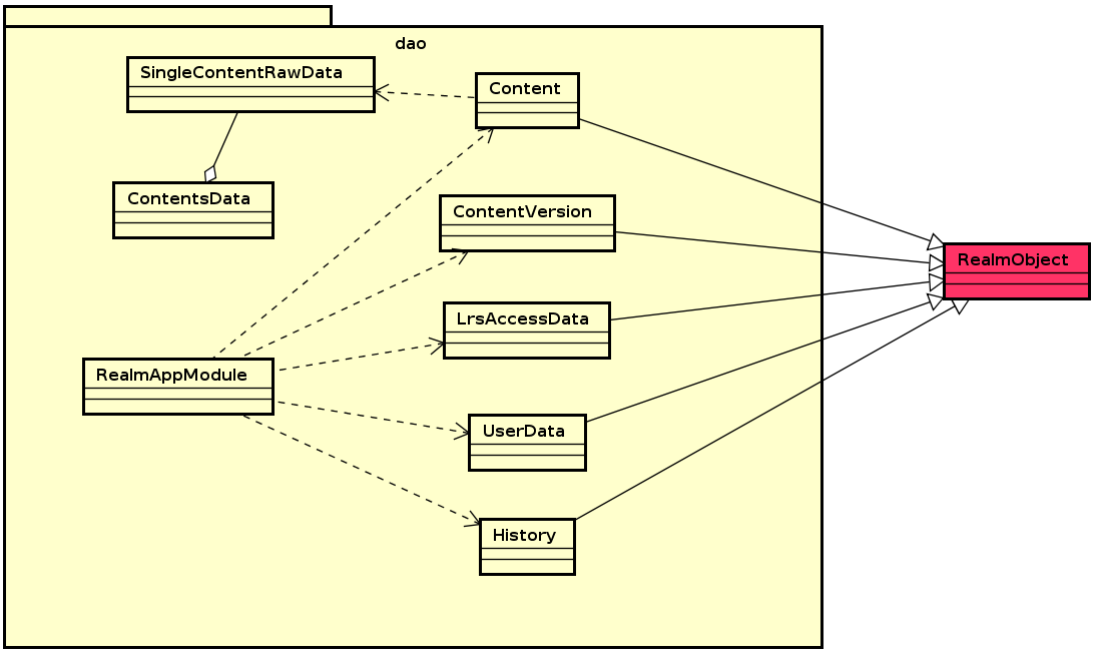

Figura 4: Package dao

Il package `dao` contiene al suo interno tutte le componenti che si occupano della rappresentazione dei dati che vengono salvati nel database interno al dispositivo. Tutte le classi al suo interno corrispondono alle tabelle del database, ad eccezione delle classi `SingleContentRawData` e `ContentsData` che sono classi utili al recupero delle informazioni necessarie per la creazione di oggetti di tipo `Content` e della classe `RealmAppModule`. Tali classi estendono tutte `RealmObject` poiché, in questo modo, possono essere salvate direttamente all'interno del database locale utilizzando la libreria Realm.
Le classi che appartengono a tale package sono:

- `Content`;
- `ContentVersion`;
- `History`;
- `UserData`;
- `LrsAccessData`;

- `SingleContentRawData`;
- `ContentsData`;
- `RealmAppModule`.

5.3 Package model::data_access

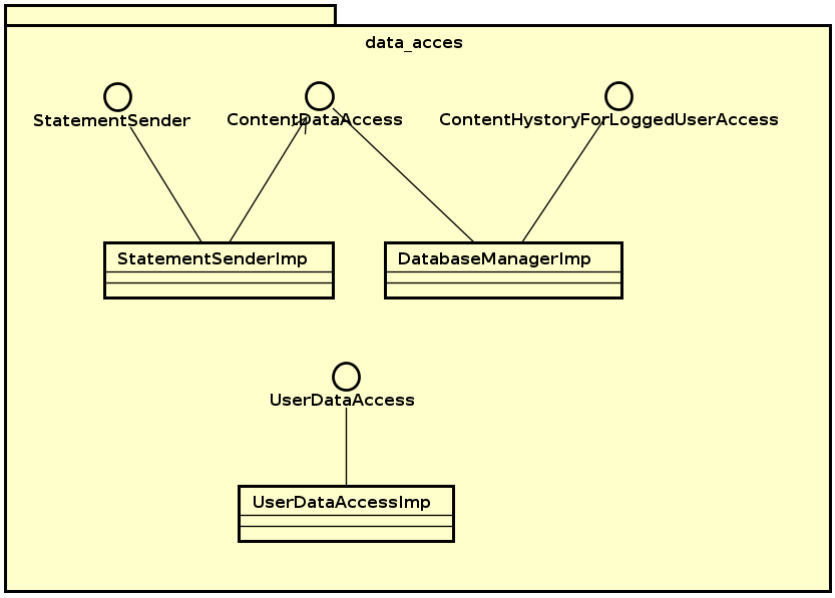

Figura 5: Package data_access

Il package `data_access` contiene al suo interno tutte le componenti che si occupano dell'accesso e alla modifica dei dati salvati nel database interno al dispositivo.
Le interfacce che appartengono a tale package sono:

- `StatementSender`;
- `ContentDataAccess`;
- `ContentHystoryForLoggedUserAccess`.

Le classi che appartengono a tale package sono:

- `DatabaseAccess`;
- `StatementSenderImp`;
- `UserDataAccessImp`.

5.4 Package model::listener

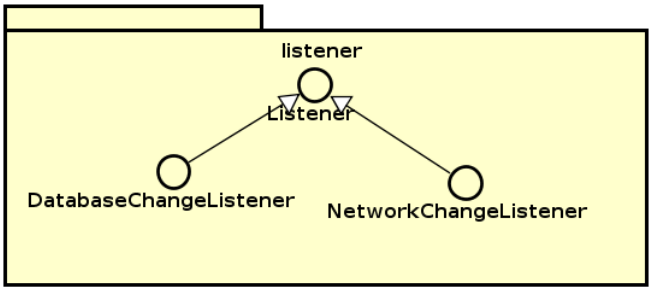

Figura 6: Package listener

Il package `listener` contiene al suo interno tutte le interfacce che devono essere implementate dalle classi che vogliono essere registrate come listener. Le interfacce che appartengono a tale package sono:

- `Listener`;
- `DatabaseChangeListener`;
- `NetworkChangeListener`.

5.5 Package model::result

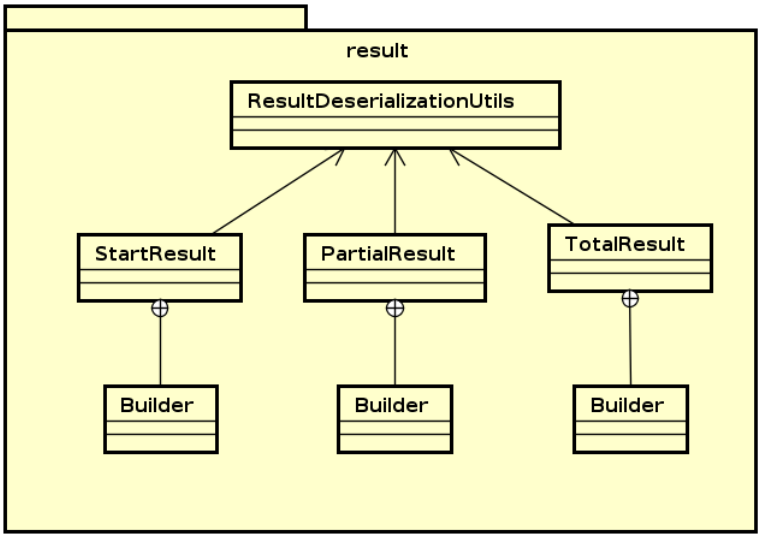

Figura 7: Package result

Il package `result` contiene al suo interno tutte le classi che si occupano della trasformazione di oggetti in formato JSON, i quali rappresentano i dati delle interazioni di un utente con un certo contenuto xAPI che devono essere registrati, in un oggetto Java.
Le classi che appartengono a tale package sono:

- `PartialResult`;
- `PartialResult.Builder`;
- `StartResult`;
- `StartResult.Builder`;
- `TotalResult`;
- `TotalResult.Builder`;
- `ResultDeserializationUtilities`.

5.6 Package model::lrs_access

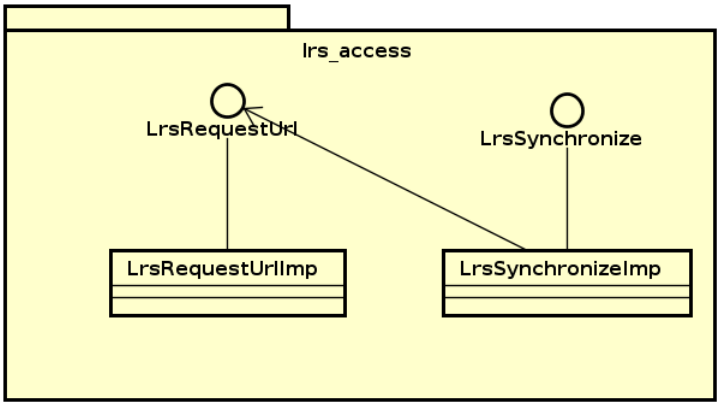

Figura 8: Package lrs_access

Il package `lrs_access` contiene al suo interno tutte le componenti che si occupano di effettuare delle richieste ad un LRS al fine di recuperare informazioni riguardanti un determinato utente.
Le interfacce che appartengono a tale package sono:

- `LrsRequestUrl`;

- `LrsSynchronize`.

Le classi che appartengono a tale package sono:

- `LrsRequestUrlImp`;
- `LrsSynchronizeImp`.

5.7 Package model::statement_object

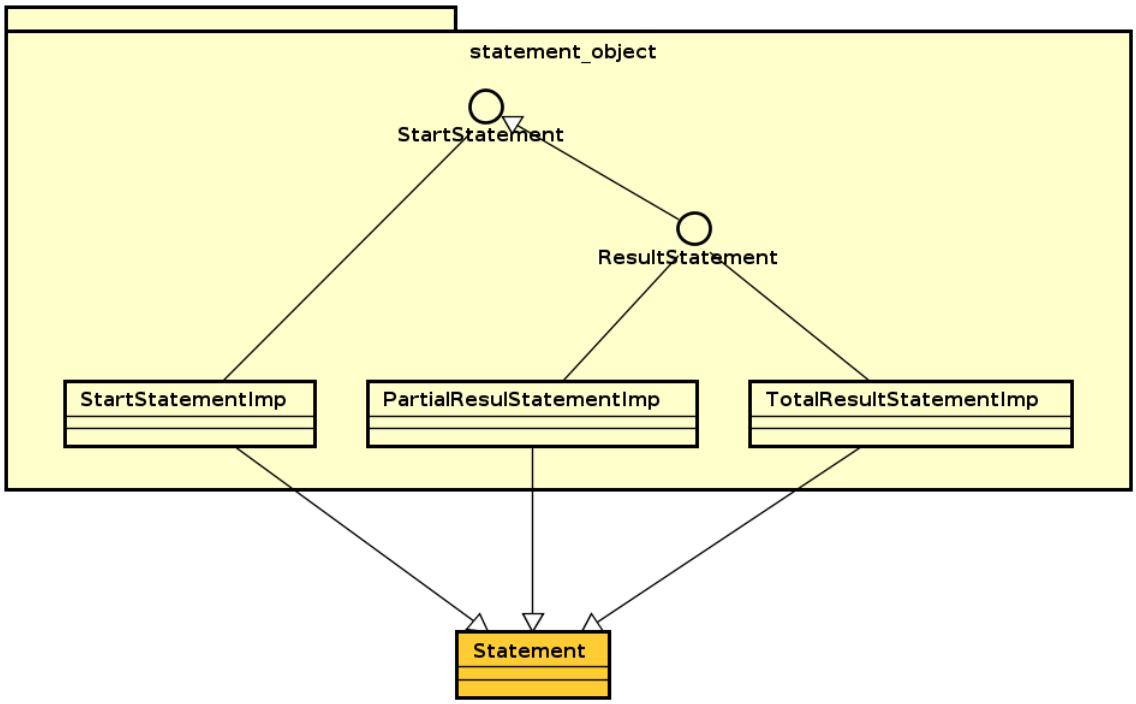

Figura 9: Package statement_object

Il package `statement_object` contiene al suo interno tutte le componenti che servono per la rappresentazione di statement xAPI che devono essere inviati ad un LRS.
Le interfacce che appartengono a tale package sono:

- `StartStatement`;
- `ResultStatement`.

Le classi che appartengono a tale package sono:

- `StartStatementImp`;
- `PartialResultStatement`;
- `TotalResultStatement`.

5.8 Package presenter

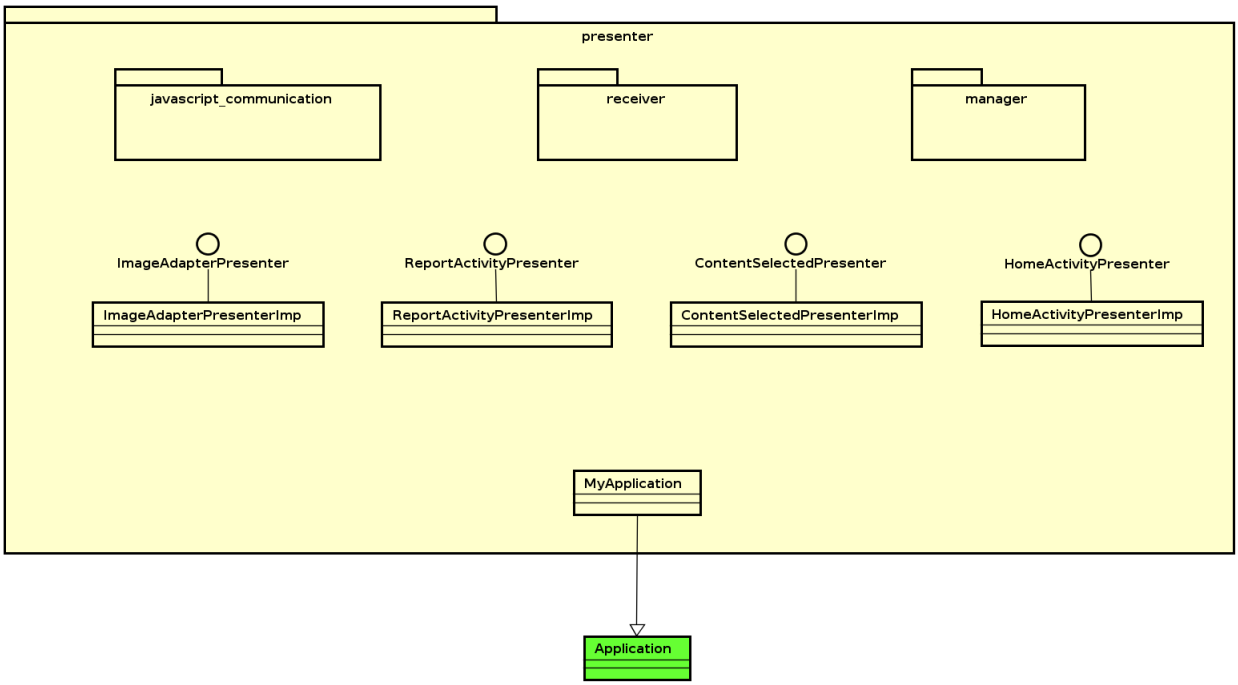

Figura 10: Package presenter

Il package `presenter` della comunicazione tra i package `model` e `view`.
I package contenuti al suo interno sono:

- `javascript_communication`;
- `manager`;
- `receiver`.

Le interfacce che appartengono a tale package sono:

- `ContentSelectedPresenter`;
- `HomeActivityPresenter`;

- `ImageAdapterPresenter`;
- `ReportActivityPresenter`.

Le classi che appartengono a tale package sono:

- `ContentSelectedPresenterImp`;
- `HomeActivityPresenterImp`;
- `ImageAdapterPresenterImp`;
- `MyApplication`;
- `ReportActivityPresenterImp`.

5.9 Package presenter::javascript_communication

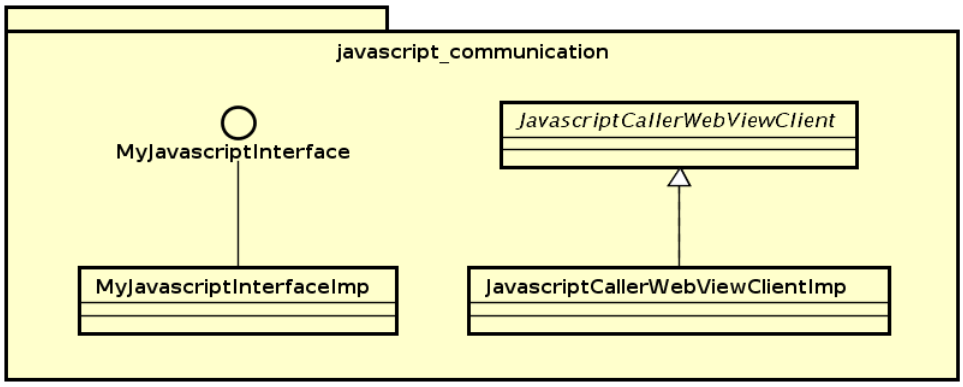

Figura 11: Package javascript_communication

Il package `javascript_communication` ha il compito di dialogare con gli script JavaScript che si occupano della gestione dei contenuti xAPI.
Le interfacce che appartengono a tale package sono:

- `MyJavascriptInterface`.

Le classi che appartengono a tale package sono:

- `JavascriptCallerWebViewClient`;
- `JavascriptCallerWebViewClientImp`;
- `MyJavascriptInterface`.

5.10 Package presenter::manager

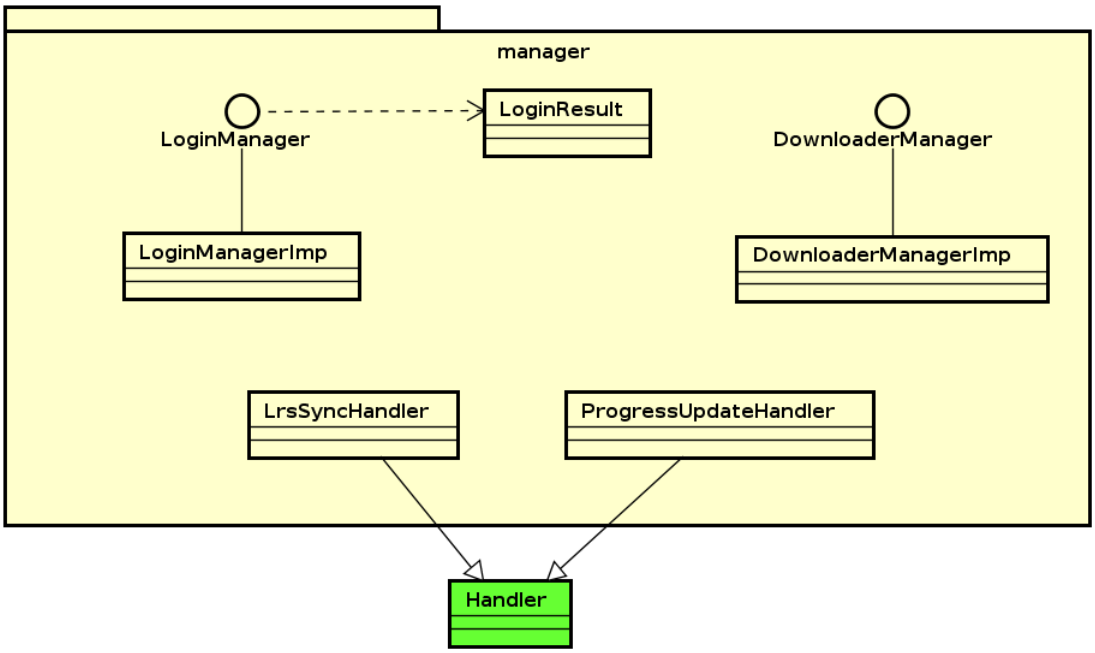

Figura 12: Package manager

Il package `manager` si occupa di effettuare delle connessioni a server per il download o per l'autenticazione e la registrazione in background e di aggiornare view e model al termine.
Le interfacce che appartengono a tale package sono:

- `DownloaderManager`;
- `LoginManager`.

Le classi che appartengono a tale package sono:

- `DownloaderManagerImp`;
- `LoginManagerImp`;
- `LoginResult`;
- `LrsSyncHandler`;
- `ProgressUpdateHandler`.

5.11 Package presenter::receiver

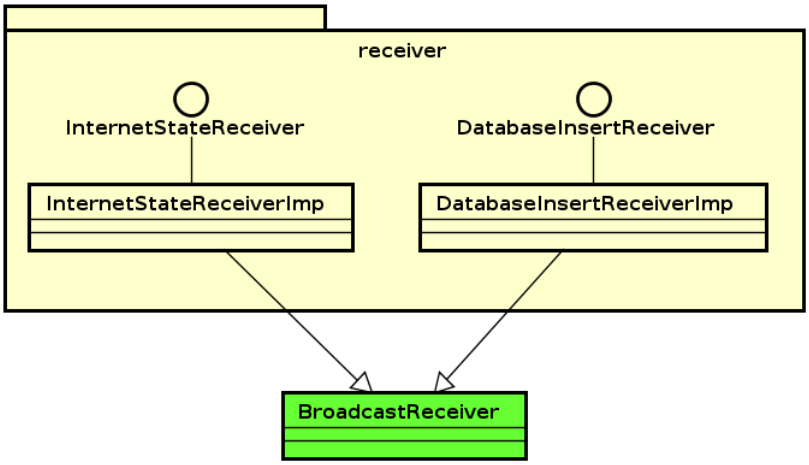

Figura 13: Package receiver

Il package `receiver` si occupa di monitorare lo stato del database offerto dalla libreria TinCanAndroid-Offline e lo stato della connessione internet del dispositivo.
Le interfacce che appartengono a tale package sono:

- `InternetStateReceiver`;
- `DatabaseInsertReceiver`.

Le classi che appartengono a tale package sono:

- `InternetStateReceiverImp`;
- `DatabaseInsertReceiverImp`.

5.12 Package view

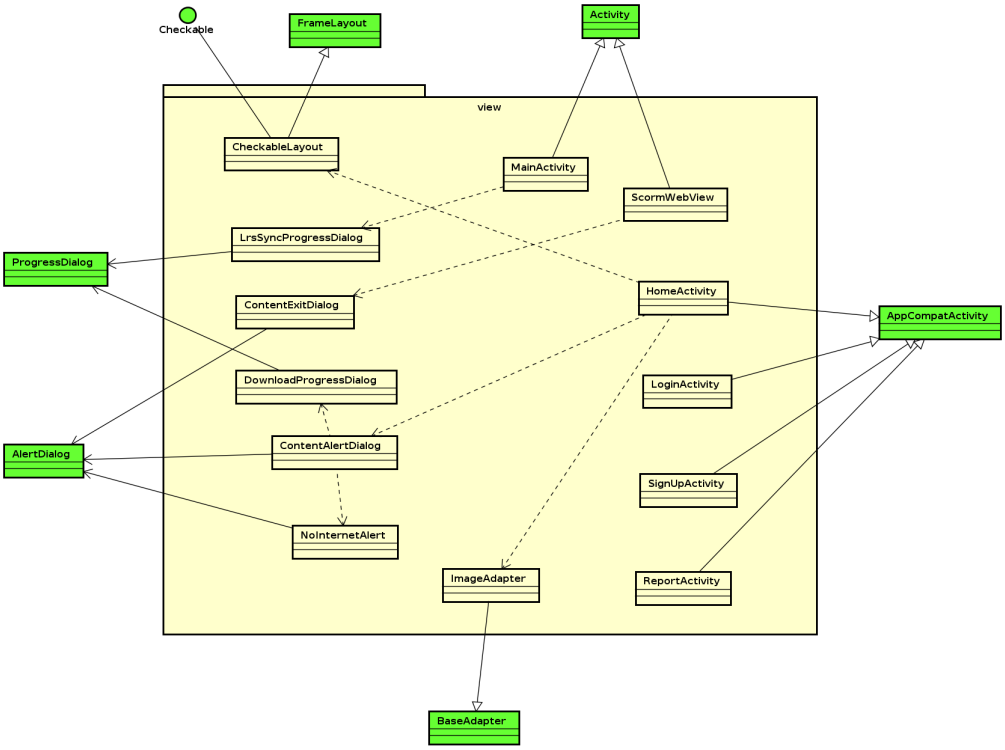

Figura 14: Package view

Il package `view` dell'interfaccia grafica dell'applicazione.
Le classi che appartengono a tale package sono:

- `CheckableLayout`;
- `ContentAlertDialog`;
- `ContentExitDialog`;
- `DownloadProgressDialog`;
- `HomeActivity`;
- `ImageAdapter`;
- `LoginActivity`;
- `LrsSyncProgressDialog`;

- `MainActivity`;
- `NoInternetAlert`;
- `ReportActivity`;
- `ScormWebView`;
- `SignUpActivity`;

5.13 Package dependency_injection

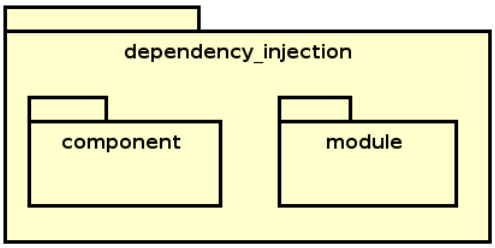

Figura 15: Package dependency_injection

Il package `dependency_injection` è il package nel quale vengono dichiarate quali classi necessitano dell'"iniezione" di qualche campo e il modo di risolvere tali dipendenze.
I package contenuti al suo interno sono:

- `component`;
- `module`.

5.14 Package dependency_injection::component

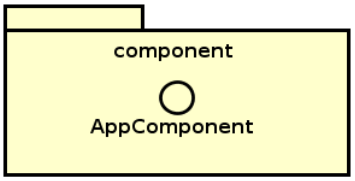

Figura 16: Package component

Il package `component` è il package nel quale vengono dichiarate quali classi necessitano dell'"iniezione" di qualche campo.
Le interfacce che appartengono a tale package sono:

- `AppComponent`.

5.15 Package dependency_injection::module

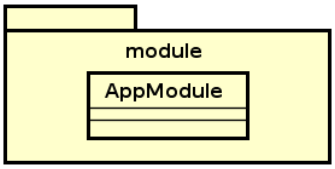

Figura 17: Package module

Il package `module` è il package nel quale viene dichiarato come risolvere le dipendenze delle classi dichiarate nel package `component`.
Le classi che appartengono a tale package sono:

- `AppModule`.