Università degli Studi di Padova

Dipartimento di matematica
Corso di Laurea in Informatica

Progettazione e sviluppo di un'applicazione mobile per l'e-learning

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 contenuti 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. Descrizione dei package: descrizione di tutti i package in cui è organizzato il codice del prodotto finale;
6. Descrizione delle classe principali: descrizione delle classi più importanti dell'applicazione;
7. Verifica e validazione: descrizione della fase di testing dell'applicazione;
8. Conclusioni: bilancio finale sull'attività di stage.

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 se si necessita solamente di permettere la fruizione di alcuni corsi non è necessario lo sviluppo di un LMS o il pagamento di una tale piattaforma ma è possibile sfruttare un qualunque spazio web per la pubblicazione dei contenuti. Per il recupero delle esperienze degli utenti, invece, è possibile utilizzare un LRS. Quest'ultimo, generalmente, non ha particolari necessità in termini di requisiti hardware della macchina su cui è installato e ne sono disponibili diversi open-source (per esempio LearningLocker). 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 formato JSON all'interno di un file 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.

Per ogni contenuto un utente po' scegliere se effettuarne il download oppure fruirne online. L'applicazione, inoltre, permette, ad un utente autenticato, di accedere ad un riepilogo delle attività riguardanti i contenuti di e-learning fruibili.

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

Identificativo	Descrizione
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 Android Studio

Android Studio è un ambiente di sviluppo integrato per lo sviluppo per la piattaforma Android. È stato annunciato il 16 maggio 2013 in occasione della conferenza Google I/O tenuta dal Product Manager Google, Katherine Chou. Android Studio è disponibile gratuitamente sotto licenza Apache 2.0.

3.2.1 Vantaggi

- creato per rispondere alle esigenze di uno sviluppatore Android;
- offre un emulatore per simulare dispositivi differenti.

3.2.2 Svantaggi

- non è intuitivo il modo di importare le librerie quando non sono disponibili in repository come Maven Central o JCenter.

3.3 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.

3.3.1 Vantaggi

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

3.3.2 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.

3.4 Espresso

Espresso è un framework per la creazione di test dell'interfaccia grafica di applicazione Android, simulando l'interazione di un utente. Tale tecnologia è stata appunto utilizzata per lo sviluppo di test che simulassero un utente che utilizza l'applicazione.

3.4.1 Vantaggi

- fornisce la sincronizzazione automatica delle azioni di test con l'interfaccia utente dell'app che si sta testando.

3.4.2 Svantaggi

- non è sempre intuitivo il modo di specificare l'oggetto grafico se deve essere effettuata un'azione.

3.5 Google AutoValue

Google AutoValue è una libreria che permette 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.

3.5.1 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.

3.5.2 Svantaggi

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

3.6 Git

Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.

3.6.1 Vantaggi

- molto veloce;
- abbastanza semplice da utilizzare.

3.6.2 Svantaggi

- in ambiente Windows dev'essere utilizzato tramite un simulatore del terminale dei sistemi Linux.

3.7 GitHub

GitHub è un sistema web di hosting per progetti che utilizzano Git per il controllo di versione.

3.7.1 Vantaggi

- permette la creazione di un account gratuito;
- permette la creazione di un numero illimitato di repository pubbliche gratuitamente;
- permette la creazione di un numero illimitato di repository private gratuitamente, se in possesso di un account studente;
- offre un sistema di gestione issue, wiki, commenti al codice e richieste di pull.

3.7.2 Svantaggi

- se non in possesso di un account studente è possibile creare le repository private solamente a pagamento.

3.8 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.

3.8.1 Vantaggi

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

3.8.2 Svantaggi

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

3.9 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.

3.9.1 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.

3.9.2 Svantaggi

- linguaggio verboso.

3.10 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 ed inviarle all'applicazione Android.

3.10.1 Vantaggi

- permette l'interazione con pagine web e con i contenuti xAPI;
- viene eseguito dal browser del client.

3.10.2 Svantaggi

- problemi di sicurezza, soprattutto in ambiente Android.

3.11 JUnit

JUnit è un framework per la creazione di unit test per il linguaggio di programmazione Java. Tale tecnologia è stata utilizzata per la creazione dei test di unità delle classi che non si occupano dell'interfaccia grafica dell'applicazione.

3.11.1 Vantaggi

- semplice creazione di test di unità, soprattutto ricorrendo alle asserzioni.

3.11.2 Svantaggi

- non permette di simulare l'interazione di un utente con l'interfaccia grafica, per effettuarne i test.

3.12 MongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti. Viene classificato come un database di tipo NoSQL, in quanto si allontana dalla struttura dei database relazionali basata su tabelle, in favore di documenti in stile JSON con schema dinamico. Ciò rende l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. Tale tecnologia è stata affrontata poichè l'applicazione comunica con un LRS che sfrutta come DBMS MongoDB, ovvero Learning Locker. Per questo è stato necessario lo studio di tale tecnologia per effettuare delle query utilizzando l'*Aggregation framework*.

3.12.1 Vantaggi

- tipi di dato flessibili;
- migliori performance in lettura/scrittura rispetto ad un database relazionale;
- scalabilità.

3.12.2 Svantaggi

- non ci sono controlli sull'integrità dei dati;
- non supporta le transazioni.

3.13 MPandroidChart

MPAndroidChart è una libreria per la creazione di grafici in Android. Essa viene usata per la creazione di grafici riassuntivi delle attività di un utente. Tale libreria fornisce anche delle funzioni di ridimensionamento, trascinamento, selezione e di animazione. È stata scelta questa libreria principalmente per le poche alternative disponibili.

3.13.1 Vantaggi

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

3.13.2 Svantaggi

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

3.14 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.

3.14.1 Vantaggi

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

3.14.2 Svantaggi

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

3.15 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. Nell'applicazione è utilizzata per la creazione e la gestione del database locale.

3.15.1 Vantaggi

- semplicità di utilizzo;
- accesso veloce ai dati;

- thread-safe;
- trasformazione automatica tra oggetti del database e oggetti Java.

3.15.2 Svantaggi

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

3.16 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.

3.16.1 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.

3.16.2 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

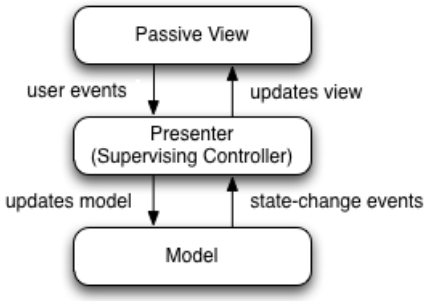

Figura 1: Rappresentazione pattern MVP

L'architettura dell'applicazione segue il pattern architetturale Model-View-Presenter, utilizzato insieme alla dependency injection. Tale scelte, unite ad un utilizzo diffuso delle interfacce, permettono la riduzione del grado di accoppiamento tra le parti del prodotto, 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, al primo avvio dell'applicazione;
- i dati relativi all'utente, quando effettua l'autenticazione o la registrazione;
- i dati di fruizione di un utente, ad ogni interazione con la slide di un corso.

4.2.1.1 Diagramma ER

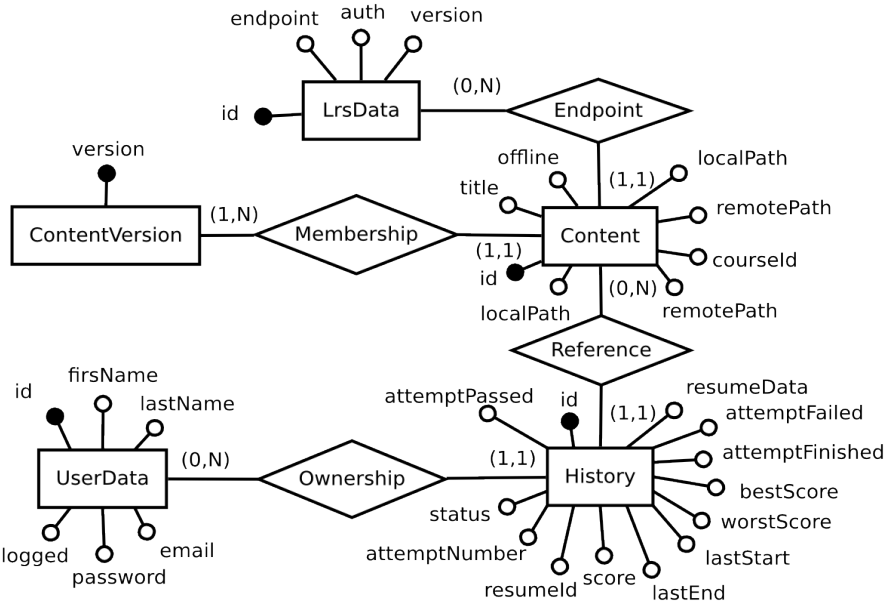

Figura 2: Diagramma ER del database locale

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 offline 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'applicazione. 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, rappresenta 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 Specifica dei contenuti

I contenuti che l'applicazione deve gestire e di cui deve permettere l'accesso agli utenti vengono, come già accennato, specificati in un file JSON che l'applicazione scarica. Tale scelta, anche se non ottimale, è stata fatta in previsione della creazione di un database che comprenda tutti i contenuti xAPI fruibili dall'applicazione. Infatti è possibile sostituire, nel file di configurazione dell'applicazione, l'indirizzo di tale file con una richiesta ad un server, il quale si occuperà di fornire all'applicazione i contenuti da mostrare agli utenti in un formato similare a quello del file attuale. Una volta effettuato il download di tale file, i contenuti al suo interno vengono utilizzati per riempire il database locale, in modo tale da potervi associare i dati di fruizione degli utenti.

4.2.3 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 l'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.

4.2.3.1 Struttura degli statement

Gli statement accettati da un LRS sono oggetti JSON che devono seguire la sintassi definita dalla specifica xAPI. Tali oggetti hanno alcuni campi obbligatori, altri opzionali che specificano le proprietà dello statement. In particolare, in questo progetto, i campi definiscono le azioni di uno specifico utente rispetto ad un determinato contenuto. Alcuni campi degli statement possono essere sia specificati da chi invia i dati,

sia dal LRS che li riceve, altri invece devono essere inseriti o da un componente o dall'altro. Di seguito sono spiegati i campi utilizzati dagli statement inviati dall'applicazione.

4.2.3.1.1 ID Il campo *ID* identifica uno statement in modo univoco all'interno di un LRS. Tale campo può essere specificato dal LRS, qualora non sia stato inserito da chi invia lo statement. Deve essere un UUID.

4.2.3.1.2 actor Il campo *actor* identifica la persona o l'oggetto a cui è associato un determinato statement. Negli statement utilizzati dall'applicativo è un oggetto composto a sua volta dai campi:

- **name**: stringa che rappresenta un nome associato alla persona o oggetto a cui è riferito lo statement;
- **mbox**: stringa che rappresenta una email nella forma "mailto:email address" che serve per l'identificazione di un singolo utente;
- **objectType**: attributo che serve per specificare che gli statement sono riferiti a singole persone o sistemi. Per questo, in tutti gli statement inviati dall'applicazione, tale campo sarà fissato uguale a "Agent".

Per l'identificazione degli utenti si è scelto di utilizzare il campo *mbox* per semplicità di gestione. Infatti la specifica xAPI permette, per identificare un utente, di associargli un generico URI oppure di specificare un oggetto *account*.

4.2.3.1.3 verb Il campo *verb* identifica l'azione compiuta dal soggetto specificato nel campo *actor*. Tale campo è un oggetto composto a sua volta dai campi:

- **id**: stringa che rappresenta un IRI che contiene il significato del verbo;
- **display**: oggetto che permette di rappresentare il verbo in varie lingue.

I verbi utilizzati nell'applicazione e il loro *id* fanno riferimento alla pagina `http://xapi.vocab.pub/datasets/adl/`, nella quale è presente una lista di verbi utilizzabili con gli statement xAPI.

4.2.3.1.4 object Il campo *object* specifica l'oggetto a cui è rivolto il verbo dichiarato nel campo *verb*. Negli statement generati dall'applicazione questo campo è utilizzato per specificare o un contenuto o una slide specifica appartenente ad un contenuto. Tale campo è un oggetto composto a sua volta dai campi:

- **objectType**: stringa che rappresenta il tipo di oggetto. Deve essere sempre uguale a "Activity se presente";
- **id**: stringa che rappresenta un identificativo del contenuto o della slide, formattata come IRI;
- **definition**: serve per specificare nome e descrizione dell'oggetto in varie lingue, oltre a specificare di che tipo di oggetto si tratta.

4.2.3.1.5 context Il campo *context* è un campo non obbligatorio che serve per specificare delle informazioni contestuali allo statement a cui appartiene. Questo campo, negli statement generati dall'applicazione, è sempre presente e specifica sempre il contenuto a cui si riferisce uno statement.

4.2.3.1.6 stored Il campo *stored* è un campo che indica il timestamp di quando è stato salvato lo statement da parte del LRS.

4.2.3.1.7 timestamp Il campo *timestamp* è un campo che indica il timestamp di quando è stata effettuata l'azione specificata dal campo *verb* da parte di un utente.

4.2.4 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, raggiungibili agli indirizzi specificati nel file di configurazione, si occupano rispettivamente del login e della registrazione, interrogando il database degli utenti e restituendo il risultato della richiesta.

4.3 Comunicazione tra Javascript e Android

Al fine di tracciare le attività degli utenti sui contenuti xAPI è stato necessario creare sia in JavaScript che in Android delle componenti che si occupassero direttamente della comunicazione tra questi linguaggi. La parte JavaScript ha il compito di recuperare le informazioni di interazione degli utenti con i contenuti e loro invio all'applicazione Android. Quest'ultima si occupa di creare gli statement dai dati che riceve e del loro invio al LRS. Le informazioni vengono recuperate da un'altra componente JavaScript, chiamata *player*, che si occupa della riproduzione dei contenuti xAPI, nei dispositivi mobili. Tale componente contiene delle variabili relative allo stato del contenuto in riproduzione, come il numero della slide, un eventuale punteggio dell'utente o quando è stato iniziato il contenuto, e si occupa di reagire all'interazione dell'utente o di mostrare la sequenza delle slide in un determinato ordine. È stato creato quindi uno script che ascolta il cambiamenti della variabile del *player* che si occupa delle slide visitate da un utente e, ad ogni variazione, raccoglie un insieme di dati e li invia all'applicazione Android. Quest'ultima si occupa di trasformare quanto ricevuto in statement. Tutto ciò è possibile riproducendo i contenuti xAPI in una *WebView* a cui è aggiunta una *JavaScriptInterface*, la quale espone dei metodi che possono essere invocati dalla componente JavaScript che si occupa del recupero delle informazioni. I dati passati tra Android e JavaScript sono degli oggetti in formato JSON.

5 Descrizione dei package

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

5.1 Package model

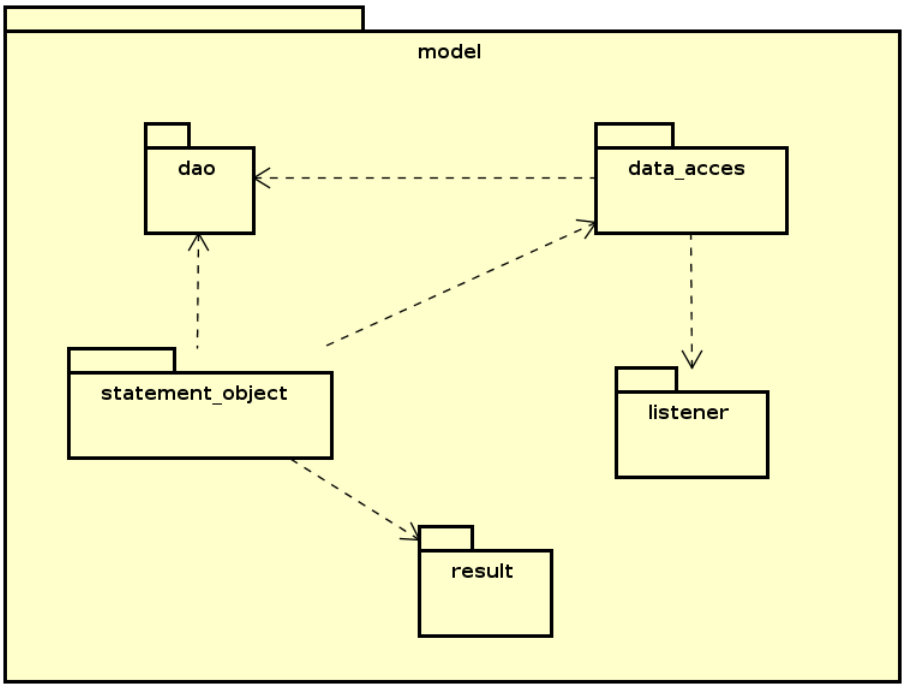

Figura 3: Package model

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

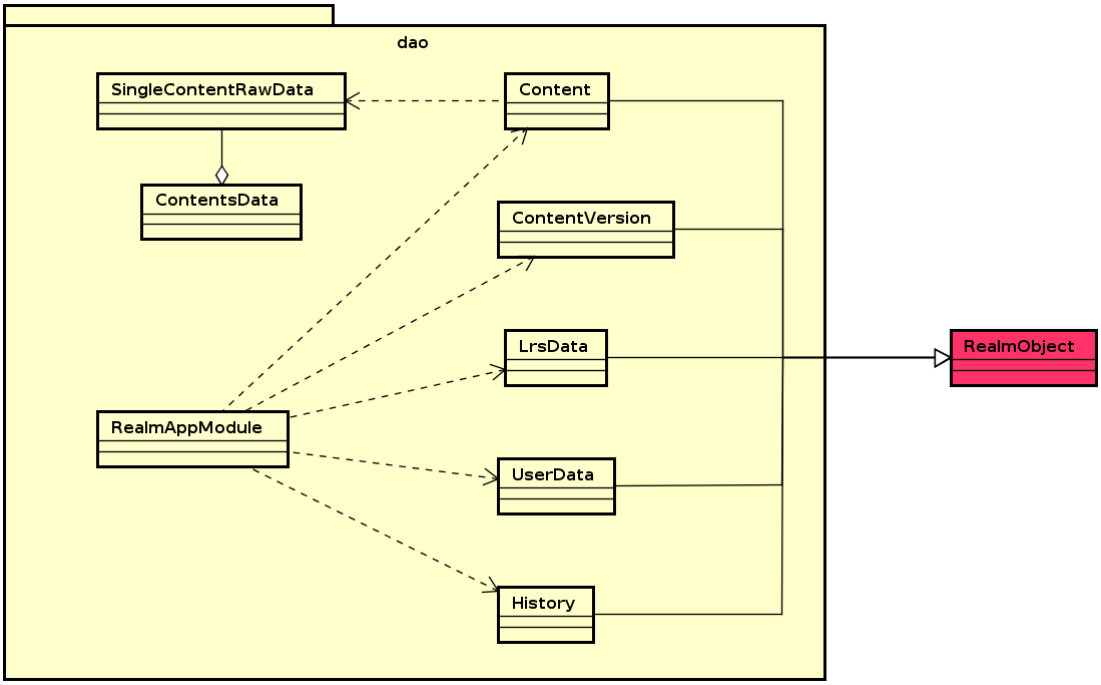

Figura 4: Package dao

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

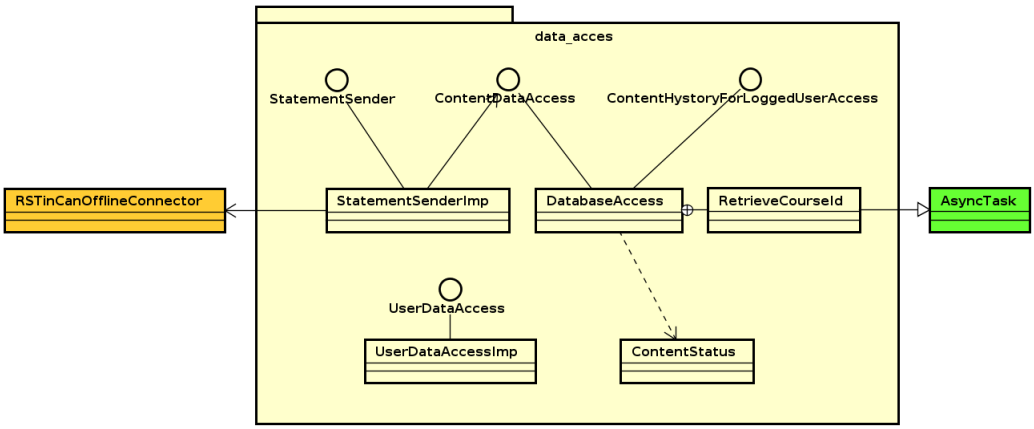

Figura 5: Package data_access

Il package `data_access` contiene al suo interno tutte le componenti che si occupano dell'accesso e della 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:

- `ContentStatus`;
- `DatabaseAccess`;
- `DatabaseAccess.RetrieveCourseId`;
- `StatementSenderImp`;
- `UserDataAccessImp`.

5.4 Package model::listener

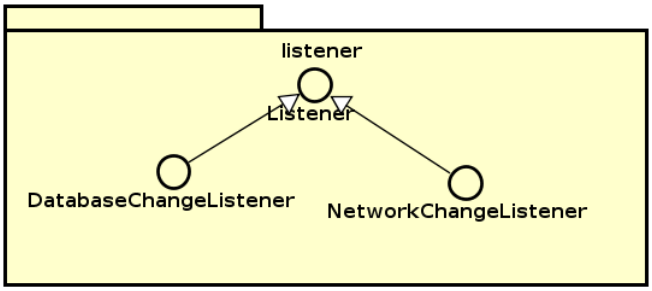

Figura 6: Package listener

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

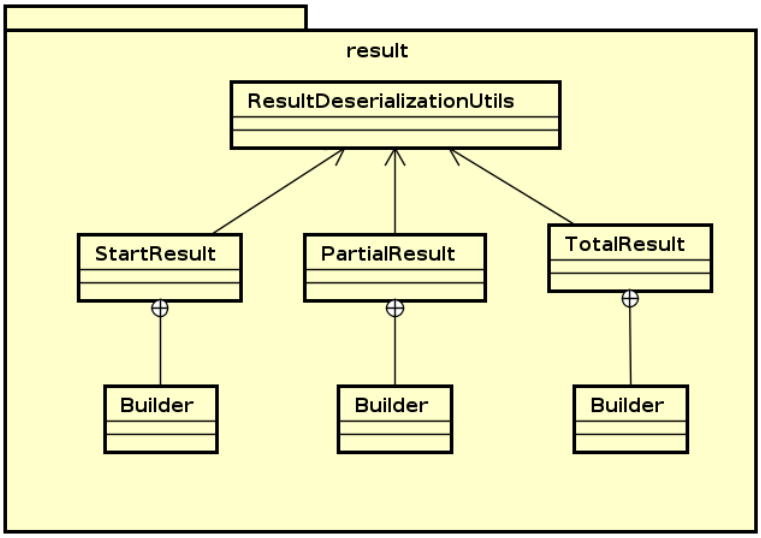

Figura 7: Package result

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

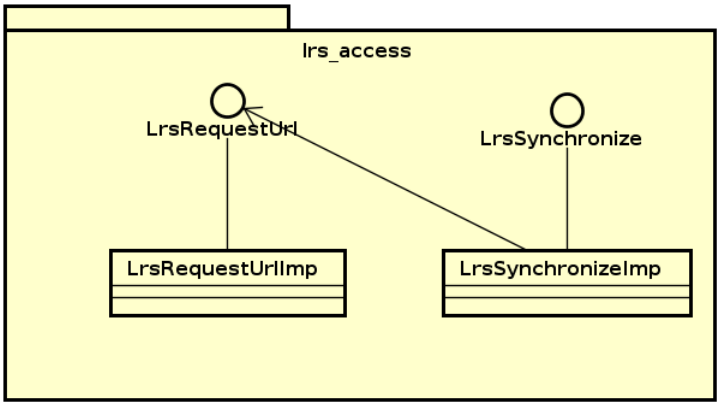

Figura 8: Package lrs_access

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

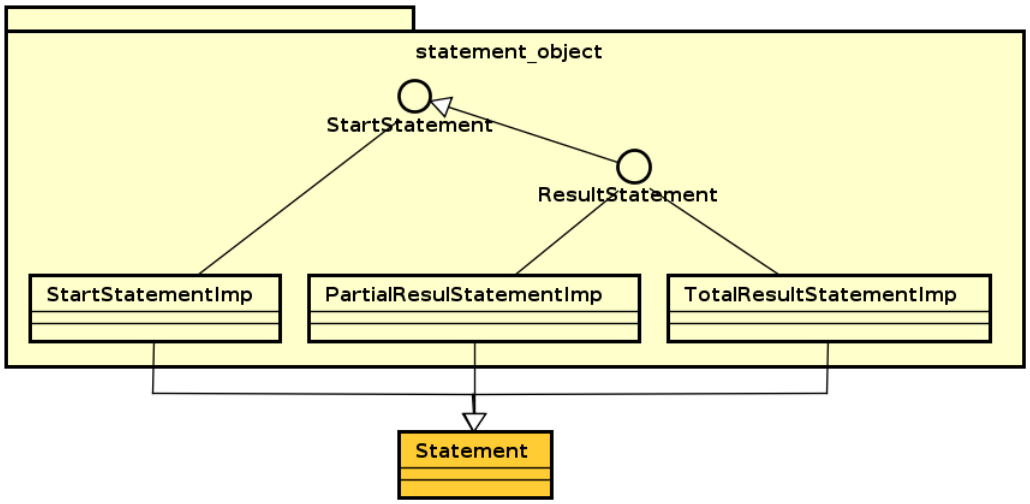

Figura 9: Package statement_object

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

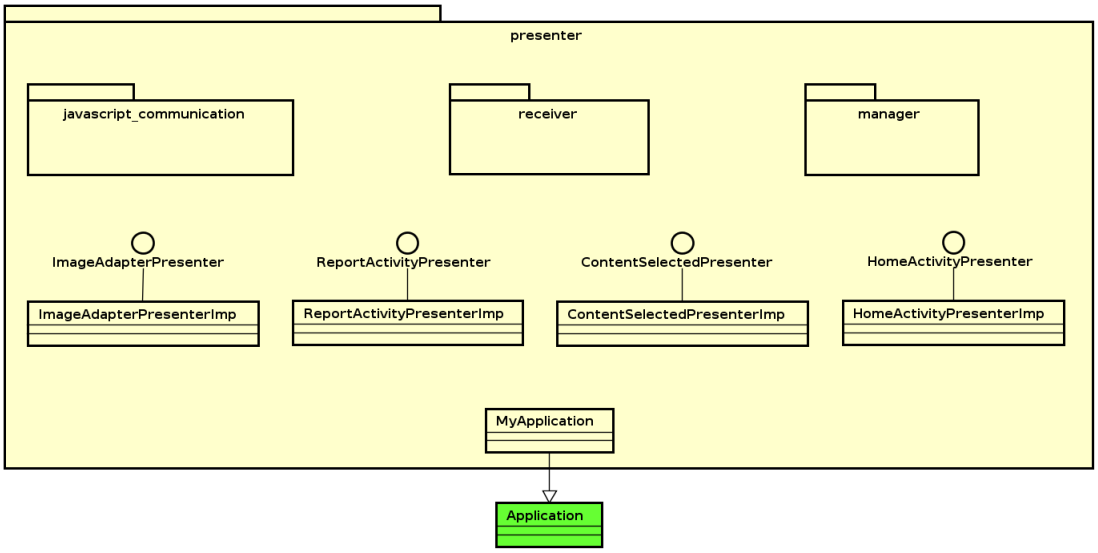

Figura 10: Package presenter

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

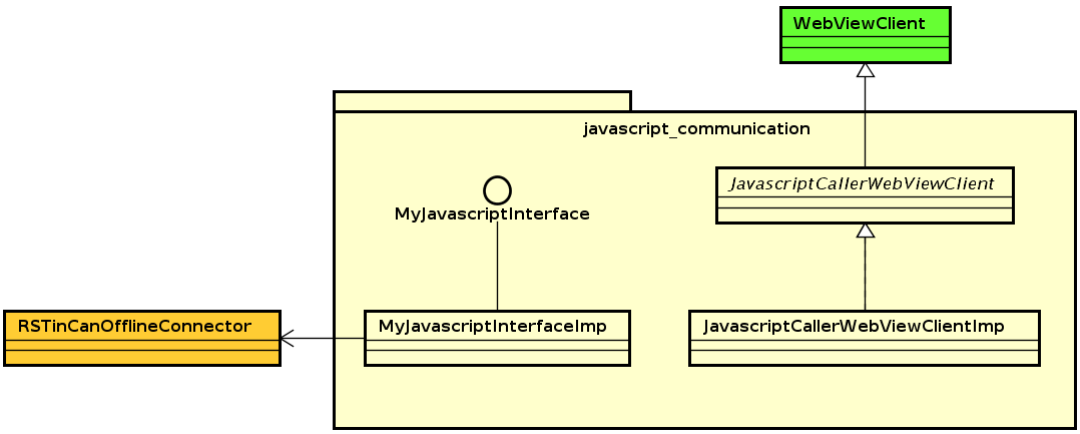

Figura 11: Package javascript_communication

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

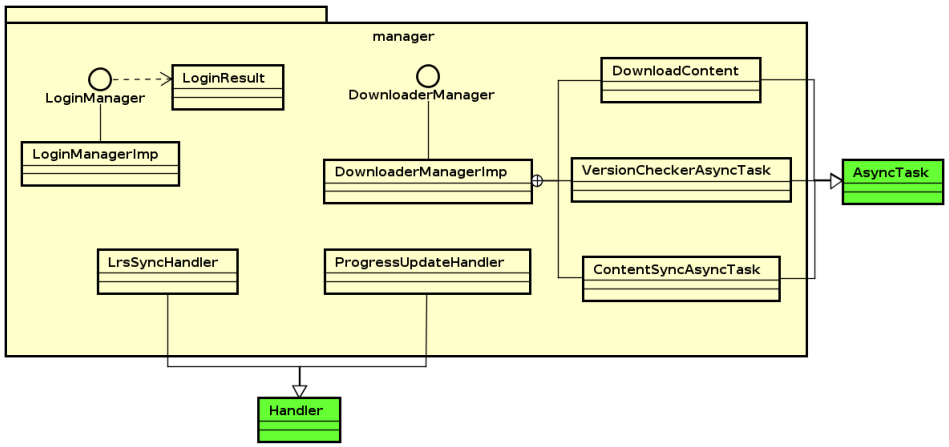

Figura 12: Package manager

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`;
- `DownloaderManagerImp.ContentSyncAsyncTask`;
- `DownloaderManagerImp.DownloadContent`;
- `DownloaderManagerImp.VersionCheckerAsyncTask`;
- `LoginManagerImp`;
- `LoginResult`;
- `LrsSyncHandler`;
- `ProgressUpdateHandler`.

5.11 Package presenter::receiver

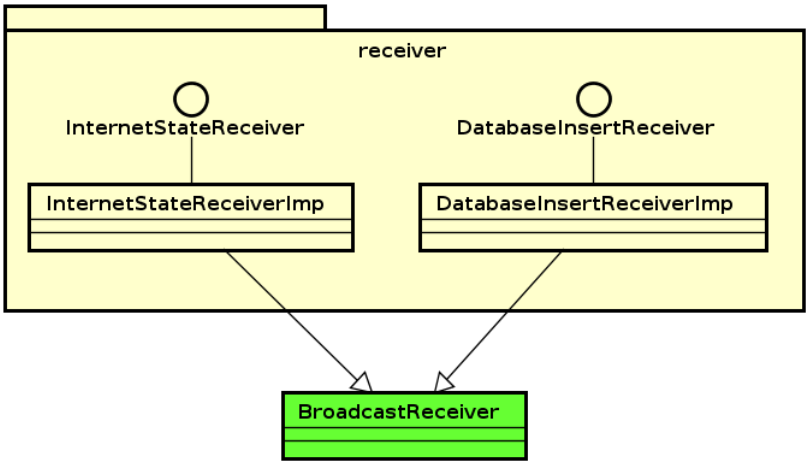

Figura 13: Package receiver

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

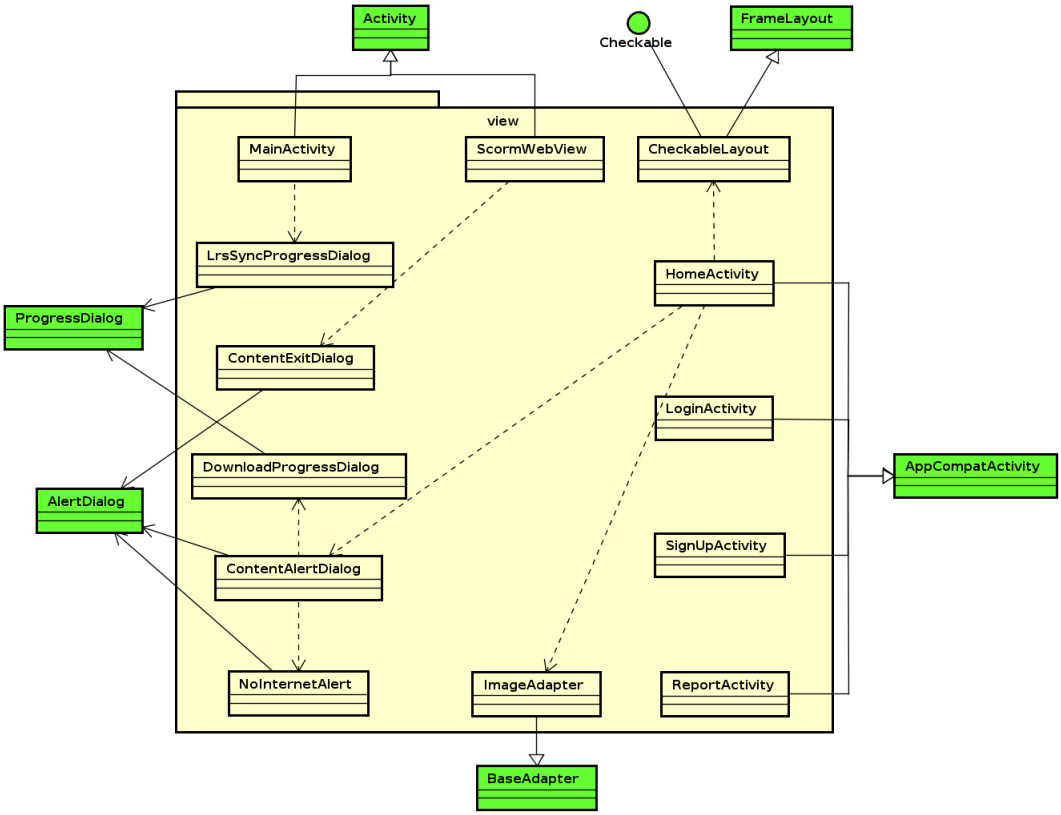

Figura 14: Package view

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

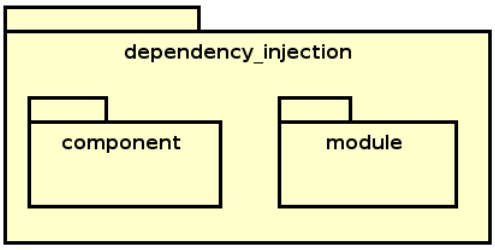

Figura 15: Package dependency_injection

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

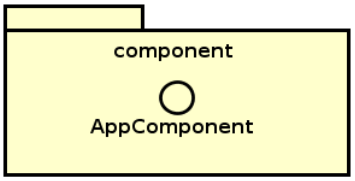

Figura 16: Package component

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

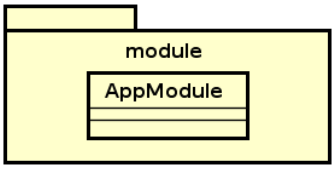

Figura 17: Package module

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`.

6 Descrizione delle classi principali

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

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 model::lrs_access::LrsSynchronizeImp

La classe `LrsSynchronizeImp` si occupa di effettuare delle richieste al LRS per 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 sfruttando 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 usate per aggiornare i dati di fruizione di un determinato utente, presenti nel database locale, utilizzando un oggetto di tipo `ContentHistoryForLoggedUserAccess`. Dal momento che è stato deciso di far uso di Learning Locker come LRS, le risposte ottenute sono in formato JSON e vengono gestite utilizzando la libreria Gson.

6.6 model::result::StartResult

La classe `StartResult` è impiegata 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 sfruttando 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`.

6.7 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 usando 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 sfruttando i campi dati dell'oggetto `StartResult`, richiesto dal costruttore della classe. Le classi `PartialResulStatementImp` e `TotalResulStatementImp` hanno compiti analoghi e sono costruite in modo simile.

6.8 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 impiegando 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.

6.9 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 di un corso. 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.

6.10 presenter::manager::ProgressUpdateHandler

La classe `ProgressUpdateHandler` si occupa dell'aggiornamento di un'istanza della classe `DownloadProgressDialog` per mostrare a video lo stato di avanzamento del download di un contenuto. Tale classe estende la classe `Handler` del framework Android. Utilizzando tale classe è possibile, oltre che impostare la percentuale di completamento del download, definire il titolo che deve essere mostrato durante il download, nascondere l'alert, mostrare al completamento del download un messaggio di errore o di download avvenuto con successo.

6.11 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.

6.12 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.

6.13 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 gestione 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`.

6.14 view::DownloadProgressDialog

La classe `DownloadProgressDialog` si occupa di mostrare una barra che indica il completamento del download di un contenuto. Tale classe offre i metodi per mostrare e nascodere l'alert che contiene la barra del completamento, impostare il livello di completamento del download e per settare il titolo del contenuto di cui si sta effettuando il download. Tale classe è utilizzata, nell'applicazione, insieme alla classe `ProgressUpdateHandler`.

6.15 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.

7 Verifica e validazione

Per eseguire la verifica e la validazione del prodotto sono stati utilizzati il framework JUnit e Espresso. Il primo strumento è un framework per lo sviluppo di unit test in Java, il secondo invece è una libreria per simulare l'interazione di un utente con l'interfaccia grafica di applicazioni Android.
L'implementazione dei test ha sia accompagnato che seguito lo sviluppo dell'applicazione. La maggior parte dei test di unità sono stati creati seguendo la metodologia test-driven development. Tale strategia prevede che la codifica dei test preceda la scrittura del codice per i quali sono stati pensati, in modo tale che ne guidino lo sviluppo.
I test di unità sviluppati coprono quasi la totalità dei metodi riguardanti le funzionalità principali dell'applicazione, mentre i test di integrazione e di sistema sono stati sviluppati in un numero inferiore, per mancanza di tempo. Per verificare il corretto funzionamento dell'applicazione è stato fatto un collaudo del prodotto insieme al tutor aziendale. Durante tale incontro sono stati inoltre discussi aspetti positivi, negativi e possibili sviluppi futuri dell'applicazione.
Tutti i test effettuati sono stati svolti su di un dispositivo Motorola Moto G 2014 con sistema operativo Android 5.2.

8 Conclusioni

8.1 Obiettivi raggiunti

In base agli obiettivi prefissati dello stage, definiti nella sezione Obiettivi, si possono trarre le seguenti conclusioni:

- gli obiettivi obbligatori, previsti dal piano di lavoro, sono stati tutti soddisfatti;
- gli obiettivi opzionali sono stati parzialmente soddisfatti.

In particolare è parzialmente soddisfatto il requisito OP1.1: l'applicazione non prevede, infatti, la possibilità di cambiare i font utilizzati. Tale funzionalità non è stata implementata per mancanza di tempo e perché è stata data priorità ai requisiti che potessero dare un maggior valore aggiunto all'applicazione.

8.2 Considerazioni sul prodotto finale

Il prodotto finale, pur soddisfacendo la maggior parte degli obiettivi prefissati, presenta alcune problematiche. Esse sono dovute principalmente alla mia scarsa esperienza, in particolar modo nel ruolo da progettista. Una delle problematiche può riguardare l'applicazione del pattern MVP: è possibile infatti che alcune componenti della *view* svolgano mansioni attribuibili al *presenter*, o viceversa. Un altro problema relativo alla progettazione dell'applicazione riguarda il non aver fissato delle metriche di qualità da seguire nello sviluppo dal codice. Pur trattandosi di un caso isolato, ciò ha permesso alla classe `DataAccess` di avere un numero piuttosto cospicuo di metodi e, di conseguenza, di statement. Negli sviluppi futuri potrebbe essere necessario ristrutturare tale classe, in modo da conferire maggiore manutenibilità al codice. Altre problematiche potrebbero riguardare l'usabilità dell'interfaccia grafica, che potrebbe non essere ottimale. Infine il prodotto dovrebbe essere testato ulteriormente. La fase di studio di fattibilità, infatti, ha richiesto qualche giorno in più di quanti erano stati preventivati nel piano di progetto e ciò ha inciso ancor di più sulla fase di test, che prevedeva, in ogni caso, un numero di giorni abbastanza limitato.

8.3 Sviluppi futuri

Alla fine dello stage, oltre a sistemare alcune problematiche evidenziate, esistono delle prospettive di miglioramento del prodotto realizzato:

- **Sviluppo di un applicazione iOS e/o Windows Phone per ampliare il bacino di utenza**: al momento il servizio è disponibile esclusivamente per dispositivi mobile con sistema operativo Android. Sarebbe meglio sviluppare applicazioni anche per gli altri sistemi operativi per dispositivi mobili, per poter ricoprire una fascia di mercato più ampia;

- **Miglioramento del tracciamento delle attività di un utente**: in questo momento l'applicazione traccia solamente le azioni di inizio e fine di un corso e i dati di fruizione delle singole slide. Potrebbe essere interessante ampliare tale funzionalità, permettendo di inviare all'LRS statement che comprendono quando un utente si è loggato, quando un utente si è registrato oppure altre azioni eseguite sull'applicazione;

- **Aggiungere la possibilità di registrarsi utilizzando i social**: questa funzionalità, anche se non vitale ai fini del prodotto, potrebbe velocizzare la fase di registrazione e autenticazione, dando la possibilità agli utenti, in versioni successive, di condividere i risultati ottenuti nei vari corsi;

- **Miglioramento del backend**: l'applicazione si appoggia su componenti esterne, le quali si occupano dell'autenticazione, registrazione e il recupero dei dati dei contenuti da visualizzare, che però non sono state sufficientemente sviluppate per mancanza di tempo. Tali componenti dovrebbero essere migliorate. Molto utile potrebbe essere, inoltre, lo sviluppo di un applicativo o di un sito web che, facendo delle query all'LRS, riuscisse a fornire delle informazioni che riguardino tutti gli utenti del prodotto. Ciò potrebbe essere molto utile per avere delle informazioni statistiche globali dell'applicazione per vedere, per esempio, quanti utenti hanno fruito di un determinato corso o quanti hanno superato una determinata domanda. Tale strumento permetterebbe infatti, oltre ad avere una panoramica sugli utenti senza andare ad interrogare con l'LRS con delle query a basso livello, di migliorare i corsi, avendo informazioni generali sulla loro fruizione.

8.4 Considerazioni finali e conoscenze acquisite

Dal punto di vista formativo l'attività di stage è stata piuttosto positiva. Ho potuto infatti approfondire la conoscenza di varie tecnologie, in particolare riguardanti il framework Android. Oltre a ciò lo stage è stato utile poichè mi ha dato l'opportunità di accrescere la mia autonomia e la capacità di farmi carico di responsabilità. Infatti il progetto è stato portato avanti solamente

da me, e, in più di un occasione, ho avuto delle problematiche, soprattutto in fase di progettazione e sviluppo, per le quali ho dovuto spendere diversi giorni per superarle. Inoltre, essendo tra le mie prime esperienze lavorative, è stato importante confrontarmi con il mondo del lavoro e mi ha permesso di capire cosa vuol dire lavorare all'interno di un azienda. D'altro canto, questo progetto non mi ha permesso di accrescere le mie capacità di lavorare in un team, come non mi ha dato l'opportunità di lavorare assieme a persone con una maggiore esperienza, che avrebbero potuto darmi degli insegnamenti importanti.
In ogni caso, le conoscenze apprese in questi tre anni di università mi hanno aiutato sia a comprendere nuove tecnologie con facilità, sia ad affrontare i problemi con metodo e flessibilità, in modo da poter trovare una soluzione.

Appendice A Esempi di statement

A.1 Statement di inizio di un contenuto

Di seguito troviamo l'esempio di uno statement che viene generato quando un utente inizia un contenuto. Tale statement può essere letto come *"Marco Zanella ha iniziato il corso Fondo Casse e Spese Minute il 23 luglio 2016 alle 18:30"*.

```
{
    "version": "1.0.0",
    "actor": {
        "objectType": "Agent",
        "name": "Marco Zanella",
        "mbox": "mailto:marco@email.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/initialized",
        "display": {
            "en-US": "initialized"
        }
    },
    "object": {
        "objectType": "Activity",
        "id": "http://Fondo%20Cassa%20e%20Spese%20Minute",
        "definition": {
            "description": {
                "und": "Fondo Cassa e Spese Minute"
            },
            "type": "http://adlnet.gov/expapi/activities/module"
        }
    },
    "context": {
        "contextActivities": {
            "parent": [
                {
                    "objectType": "Activity",
                    "id": "http://Fondo%20Cassa%20e%20Spese%20
    Minute"
                }
            ],
            "grouping": [
                {
                    "objectType": "Activity",
                    "id": "http://Fondo%20Cassa%20e%20Spese%20
    Minute"
                }
            ]
```

```
        }
    },
    "id": "cef4834e-6dc5-4c01-8979-e0222dc8a100",
    "authority": {
        "objectType": "Agent",
        "name": "New Client",
        "mbox": "mailto:hello@learninglocker.net"
    },
    "stored": "2016-07-23T18:30:06.791800+02:00",
    "timestamp": "2016-07-23T18:30:06.791800+02:00"
}
```


A.2 Statement di visualizzazione di una slide di un contentuto

Di seguito troviamo l'esempio di uno statement che viene generato quando un utente visualizza una slide che non prevede un test. Tale statement può essere letto come *"Marco Zanella ha fatto pratica sulla slide Miroglio Fashion del corso Fondo Casse e Spese Minute il 23 luglio 2016 alle 18:30"*.


```
{
    "version": "1.0.0",
    "actor": {
        "objectType": "Agent",
        "name": "Marco Zanella",
        "mbox": "mailto:marco@email.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/experienced",
        "display": {
            "en-US": "experienced"
        }
    },
    "object": {
        "objectType": "Activity",
        "id": "http://Fondo%20Cassa%20e%20Spese%20Minute/6
    KswNmNiPKX",
        "definition": {
            "name": {
                "en-US": "Miroglio Fashion"
            },
            "description": {
                "en-US": "Fondo Cassa e Spese Minute"
            },
```

```
            "type": "http://adlnet.gov/expapi/activities/module"
        }
    },
    "context": {
        "contextActivities": {
            "parent": [
                {
                    "objectType": "Activity",
                    "id": "http://Fondo%20Cassa%20e%20Spese%20
    Minute"
                }
            ],
            "grouping": [
                {
                    "objectType": "Activity",
                    "id": "http://Fondo%20Cassa%20e%20Spese%20
    Minute"
                }
            ]
        }
    },
    "id": "5f393687-79ca-4f0b-bcd9-7014fa37a984",
    "authority": {
        "objectType": "Agent",
        "name": "New Client",
        "mbox": "mailto:hello@learninglocker.net"
    },
    "stored": "2016-07-23T18:30:07.820700+02:00",
    "timestamp": "2016-07-23T18:30:07.820700+02:00"
}
```


A.3 Statement di risposta ad una domanda di un test

Di seguito troviamo l'esempio di uno statement che viene generato quando un utente risponde ad una slide che prevede un test. Tale statement può essere letto come *"Marco Zanella ha risposto alla domanda Questo è un vero o falso del corso DEMO_ TEST il 23 luglio 2016 alle 11:26"*.


```
{
    "version": "1.0.0",
    "actor": {
        "objectType": "Agent",
        "name": "Marco Zanella",
        "mbox": "mailto:marco@email.com"
    },
```

```
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/answered",
        "display": {
            "en-US": "answered"
        }
    },
    "object": {
        "objectType": "Activity",
        "id": "http://6PdqP7rkRtE_course_id/618Dl3BgPcO",
        "definition": {
            "name": {
                "en-US": "Questo \u00e8 un vero o falso %
    Variable1%"
            },
            "description": {
                "en-US": "DEMO_TEST"
            },
            "type": "http://adlnet.gov/expapi/activities/module"
        }
    },
    "result": {
        "score": {
            "scaled": 0.3333,
            "raw": 10,
            "max": 10
        },
        "success": true,
        "response": "true"
    },
    "context": {
        "contextActivities": {
            "parent": [
                {
                    "objectType": "Activity",
                    "id": "http://6PdqP7rkRtE_course_id"
                }
            ],
            "grouping": [
                {
                    "objectType": "Activity",
                    "id": "http://6PdqP7rkRtE_course_id"
                }
            ]
        }
    },
    "id": "c7f63cf1-54a4-45d3-b5e2-06e5d2561860",
    "authority": {
        "objectType": "Agent",
        "name": "New Client",
```

```
        "mbox": "mailto:hello@learninglocker.net"
    },
    "stored": "2016-07-23T11:26:29.352200+02:00",
    "timestamp": "2016-07-23T11:26:29.352200+02:00"
}
```

A.4 Statement finale di un contenuto che comprende dei test

Di seguito troviamo l'esempio di uno statement che viene generato quando un utente termina un corso che comprende un test. Tale statement può essere letto come *"Marco Zanella ha termianto il corso DEMO_TEST, con il punteggio di 20/30, non superando il corso, il 23 luglio 2016 alle 11:26"*.

```
{
    "version": "1.0.0",
    "actor": {
        "objectType": "Agent",
        "name": "Marco Zanella",
        "mbox": "mailto:marco@email.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/terminated",
        "display": {
            "en-US": "terminated"
        }
    },
    "object": {
        "objectType": "Activity",
        "id": "http://6PdqP7rkRtE_course_id",
        "definition": {
            "description": {
                "und": "DEMO TEST"
            },
            "type": "http://adlnet.gov/expapi/activities/module"
        }
    },
    "result": {
        "score": {
            "scaled": 0.6666,
            "raw": 20,
            "max": 30
        },
        "success": false
```

```
    },
    "context": {
        "contextActivities": {
            "parent": [
                {
                    "objectType": "Activity",
                    "id": "http://6PdqP7rkRtE_course_id"
                }
            ],
            "grouping": [
                {
                    "objectType": "Activity",
                    "id": "http://6PdqP7rkRtE_course_id"
                }
            ]
        }
    },
    "id": "f687a812-587d-4159-bbd8-b53f44ed34ca",
    "authority": {
        "objectType": "Agent",
        "name": "New Client",
        "mbox": "mailto:hello@learninglocker.net"
    },
    "stored": "2016-07-23T11:26:38.193700+02:00",
    "timestamp": "2016-07-23T11:26:38.193700+02:00"
}
```

Appendice B Esempi di richieste all'LRS

B.1 Richiesta per ottenere l'ultimo timestamp di inizio per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere l'ultimo timestamp di inizio per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica l'ultimo contenuto iniziato nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
        "$match":{
            "statement.actor.mbox":"mailto:marco@email.com",
            "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    initialized"
        }
    },
    {
        "$sort":{
            "statement.timestamp":-1
        }
    },
    {
        "$group":{
            "_id":"$statement.object.id",
            "_timestamp":{
                "$max":"$statement.timestamp"
            }
        }
    },
    {
        "$project":{
            "_id":"$_id",
            "_timestamp":1
        }
    }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
```

```
      "_id": "http://6mBlxaNrSNf_course_id",
      "_timestamp": "2016-07-19T22:05:53.049400+02:00"
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "_timestamp": "2016-07-21T14:11:25.970000+02:00"
    },
    {
      "_id": "http://5nJwugn0dFZ_course_id",
      "_timestamp": "2016-07-21T16:20:21.906800+02:00"
    }
  ],
  "ok": 1
}
```


B.2 Richiesta per ottenere l'ultimo timestamp di fine per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere l'ultimo timestamp di fine per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica l'ultimo contenuto terminato nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
   {
      "$match":{
         "statement.actor.mbox":"mailto:marco@email.com",
         "statement.verb.id":"http://adlnet.gov/expapi/verbs/
   terminated"
      }
   },
   {
      "$sort":{
         "statement.timestamp":-1
      }
   },
   {
      "$group":{
         "_id":"$statement.object.id",
         "_timestamp":{
            "$max":"$statement.timestamp"
```

```
            },
            "_result":{
               "$first":"$statement.result"
            }
         }
      },
      {
         "$project":{
            "_id":"$_id",
            "_timestamp":1,
            "_result":1
         }
      }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "_timestamp": "2016-07-19T22:07:30.904800+02:00",
      "_result": {
        "score": {
          "scaled": 1,
          "raw": 4,
          "max": 4
        },
        "success": true
      }
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "_timestamp": "2016-07-20T07:50:31.215900+02:00",
      "_result": {
        "score": {
          "scaled": 0.6014,
          "raw": 421,
          "max": 700
        },
        "success": false
      }
    }
  ],
  "ok": 1
}
```

B.3 Richiesta per ottenere il numero di tentativi iniziati per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il numero di tentativi iniziati per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il numero di tentativi iniziati per ciascun contenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
        "$match":{
            "statement.actor.mbox":"mailto:marco@email.com",
            "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    initialized"
        }
    },
    {
        "$group":{
            "_id":"$statement.object.id",
            "count":{
                "$sum":1
            }
        }
    }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://5nJwugn0dFZ_course_id",
      "count": 18
    },
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "count": 3
    },
    {
```

```
      "_id": "http://6PdqP7rkRtE_course_id",
      "count": 19
    }
  ],
  "ok": 1
}
```

B.4 Richiesta per ottenere il numero di tentativi terminati per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il numero di tentativi terminati per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il numero di tentativi terminati per ciascun contenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
   {
      "$match":{
         "statement.actor.mbox":"mailto:marco@email.com",
         "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    terminated"
      }
   },
   {
      "$group":{
         "_id":"$statement.object.id",
         "count":{
            "$sum":1
         }
      }
   }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "count": 2
```

```
    },
    {
      "_id": "http://6PdqP7rkRtE_course_id",
      "count": 11
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "count": 7
    }
  ],
  "ok": 1
}
```

B.5 Richiesta per ottenere il numero di tentativi superati per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il numero di tentativi superati per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il numero di tentativi superati per ciascun contenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
       "$match":{
          "statement.actor.mbox":"mailto:marco@email.com",
          "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    terminated",
          "statement.result.success":true
       }
    },
    {
       "$group":{
          "_id":"$statement.object.id",
          "count":{
             "$sum":1
          }
       }
    }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "count": 2
    },
    {
      "_id": "http://6PdqP7rkRtE_course_id",
      "count": 4
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "count": 2
    }
  ],
  "ok": 1
}
```

B.6 Richiesta per ottenere il numero di tentativi non superati per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il numero di tentativi non superati per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il numero di tentativi non superati per ciascun contenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
        "$match":{
            "statement.actor.mbox":"mailto:marco@email.com",
            "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    terminated",
            "statement.result.success":false
        }
    },
    {
        "$group":{
            "_id":"$statement.object.id",
            "count":{
                "$sum":1
            }
```

```
        }
    }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6PdqP7rkRtE_course_id",
      "count": 7
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "count": 5
    }
  ],
  "ok": 1
}
```

B.7 Richiesta per ottenere il miglior punteggio per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il miglior punteggio per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il miglior punteggio ottenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
        "$match":{
            "statement.actor.mbox":"mailto:marco@email.com",
            "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    terminated"
        }
    },
    {
        "$group":{
            "_id":"$statement.object.id",
            "_bestScore":{
```

```
            "$max":"$statement.result.score.scaled"
        }
    }
  }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "_bestScore": 1
    },
    {
      "_id": "http://6PdqP7rkRtE_course_id",
      "_bestScore": 1
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "_bestScore": 0.8585
    }
  ],
  "ok": 1
}
```

B.8 Richiesta per ottenere il peggior punteggio per ciascun contenuto per un utente

Di seguito troviamo l'esempio di una richiesta all'LRS per ottenere il peggior punteggio per ciascun contenuto, per un utente specifico. Tale richiesta è utilizzata dall'applicazione per tenere aggiornato il campo che indica il peggior punteggio ottenuto nella sezione che mostra il report dei dati di fruizione di un utente.

```
http://lrs.modonetwork.com/public/api/v1/statements/aggregate?
    pipeline=[
    {
        "$match":{
            "statement.actor.mbox":"mailto:marco@email.com",
            "statement.verb.id":"http://adlnet.gov/expapi/verbs/
    terminated"
```

```
        }
    },
    {
        "$group":{
            "_id":"$statement.object.id",
            "_worstScore":{
                "$min":"$statement.result.score.scaled"
            }
        }
    }
]
```

Di seguito troviamo l'esempio di una risposta dell'LRS.

```
{
  "result": [
    {
      "_id": "http://6mBlxaNrSNf_course_id",
      "_worstScore": 1
    },
    {
      "_id": "http://6PdqP7rkRtE_course_id",
      "_worstScore": 0.3333
    },
    {
      "_id": "http://6bhYi83IsPS_course_id",
      "_worstScore": 0.19
    }
  ],
  "ok": 1
}
```

Glossario

Annotazione Modo per aggiungere metadati nel codice sorgente Java che possono essere disponibili al programmatore durante l'esecuzione.

Dependency injection Design pattern il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità del software, diminuendo le dipendenze tra le varie componenti che lo compongono.

Design pattern Soluzione progettuale generale ad un problema ricorrente.

E-learning Utilizzo di tecnologie multimediali e di Internet per migliorare la qualità dell'apprendimento facilitando l'accesso alle risorse e ai servizi, così come anche agli scambi in remoto e alla collaborazione.

Experience API Specifica software per l'e-learning che definisce il modo in cui i contenuti e il sistema che li eroga devono dialogare. Le esperienze di apprendimento sono registrate in un Learning Record Store che può essere interno o esterno ad un Learning Management System.

IRI Acronimo di Internationalized Resource Identifier, forma generale di Uniform Resource Identifier costituita, a differenza di una URI, da una sequenza di caratteri appartenenti all'Universal Character Set (Unicode/ISO 10646), e ciò significa che al suo interno possono occorrere caratteri non appartenenti all'insieme ASCII.

JSON Acronimo di JavaScript Object Notation, formato adatto all'interscambio di dati fra applicazioni client-server. È basato sul linguaggio JavaScript Standard ECMA-262 3ª edizione dicembre 1999, ma ne è indipendente.

Learning Management System Piattaforma applicativa (o insieme di programmi) che permette l'erogazione dei corsi in modalità e-learning al fine di contribuire a realizzare le finalità previste dal progetto educativo dell'istituzione proponente. Il learning management system presidia la distribuzione dei corsi on-line, l'iscrizione degli studenti e il tracciamento delle attività on-line.

Learning Record Store Repository utilizzato per salvare in modo permanente le esperienze di fruizione di corsi di e-learning che seguono la specifica Experience API.

LMS Acronimo di Learning Management System.

LRS Acronimo di Learning Record Store.

SCORM Acronimo di Shareable Content Object Reference Model, è una raccolta di specifiche tecniche che consente, primariamente, lo scambio di contenuti digitali in maniera indipendente dalla piattaforma. Nell'e-Learning definisce le specifiche relative al riutilizzo, tracciamento e catalogazione degli oggetti didattici, strutturati i corsi.

Statement Nel contesto dell'e-learning, dichiarazione utilizzata per il tracciamento di una attività di apprendimento.

Tap Breve tocco sullo schermo, solitamente utilizzato per la selezione di un elemento.

Tin Can API Vedi Experience API.

URI Acronimo di Internationalized Resource Identifier, stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, ecc. e la rende disponibile tramite protocolli come HTTP o FTP.

xAPI Vedi Experience API.

Riferimenti bibliografici

[1] Android Developers: https://developer.android.com

[2] Documentazione online di MongoDB riguardante la'ggregation framework: https://docs.mongodb.com/manual/aggregation

[3] Tin Can API: https://tincanapi.com

[4] Specifica xAPI: https://github.com/adlnet/xAPI-Spec/blob/master/xAPI.md

[5] Documentazione Learning Locker: http://docs.learninglocker.net

[6] Vocabolario per gli statement: http://xapi.vocab.pub/datasets/adl

[7] Wikipedia: https://it.wikipedia.org