UNIBOOK

RELAZIONE ANALISI E SVILUPPO

Componenti del gruppo: Luca Campana, Antonio Scandurra, Andrea Zamberletti

UNIBOOK

RELAZIONE ANALISI E SVILUPPO

ANALISI DEL PROBLEMA

Considerata l'esigenza di gestire in modo semplice la vita accademica di uno studente universitario, abbiamo deciso di sviluppare un'applicazione Android atta a gestire questo problema.

Nello specifico volevamo che l'applicazione offrisse le seguenti funzionalità:

- Memorizzare e visualizzare i corsi frequentati (e le relative informazioni, e.g. il numero dei crediti, il nome del docente)
- Tenere traccia degli esami sostenuti annettendo i risultati ottenuti
- Calcolare la media relativa all'andamento della carriera e ipotizzarne il cambiamento attraverso la previsione del risultato di un esame futuro
- Calcolare la base del voto di laurea
- Pianificare lo svolgimento di un esame ricevendo così un promemoria due settimane prima dell'appello
- Visualizzare l'orario delle lezioni

PROGETTAZIONE ARCHITETTURALE

In questa fase è stata effettuata la progettazione dell'applicativo, focalizzandosi in particolare sullo studio e l'organizzazione delle varie entità introdotte e delle loro relazioni.

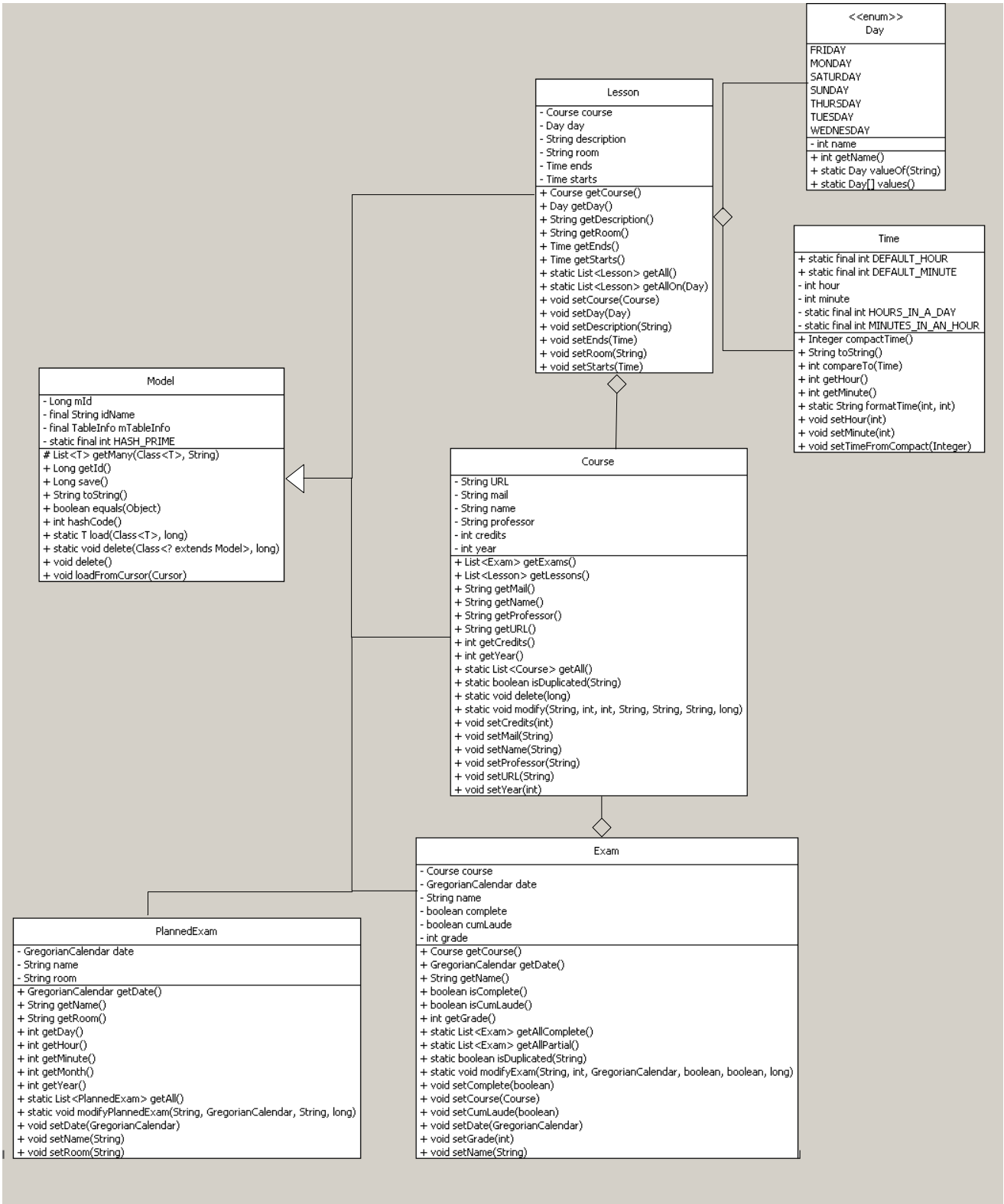
La fase di progettazione è stata guidata da un uso consistente delle linee guida Google per lo sviluppo di applicazioni Android. Queste infatti suggeriscono di suddividere la parte di presentazione (Fragment), quella di interazione (Activity) e quella di modello. Per quanto riguarda la persistenza, ci siamo avvalsi del pattern ActiveRecord, in quanto il nostro modello combaciava perfettamente con la struttura su database.

Il disaccoppiamento derivante dalla suddetta organizzazione ha consentito di sviluppare le varie funzionalità in maniera semplice, grazie all'estrema riusabilità dei componenti.

ACTIVEANDROID

È la libreria più diffusa per usufruire del pattern ActiveRecord su applicazioni Android. Per ulteriori dettagli visitare www.activeandroid.com

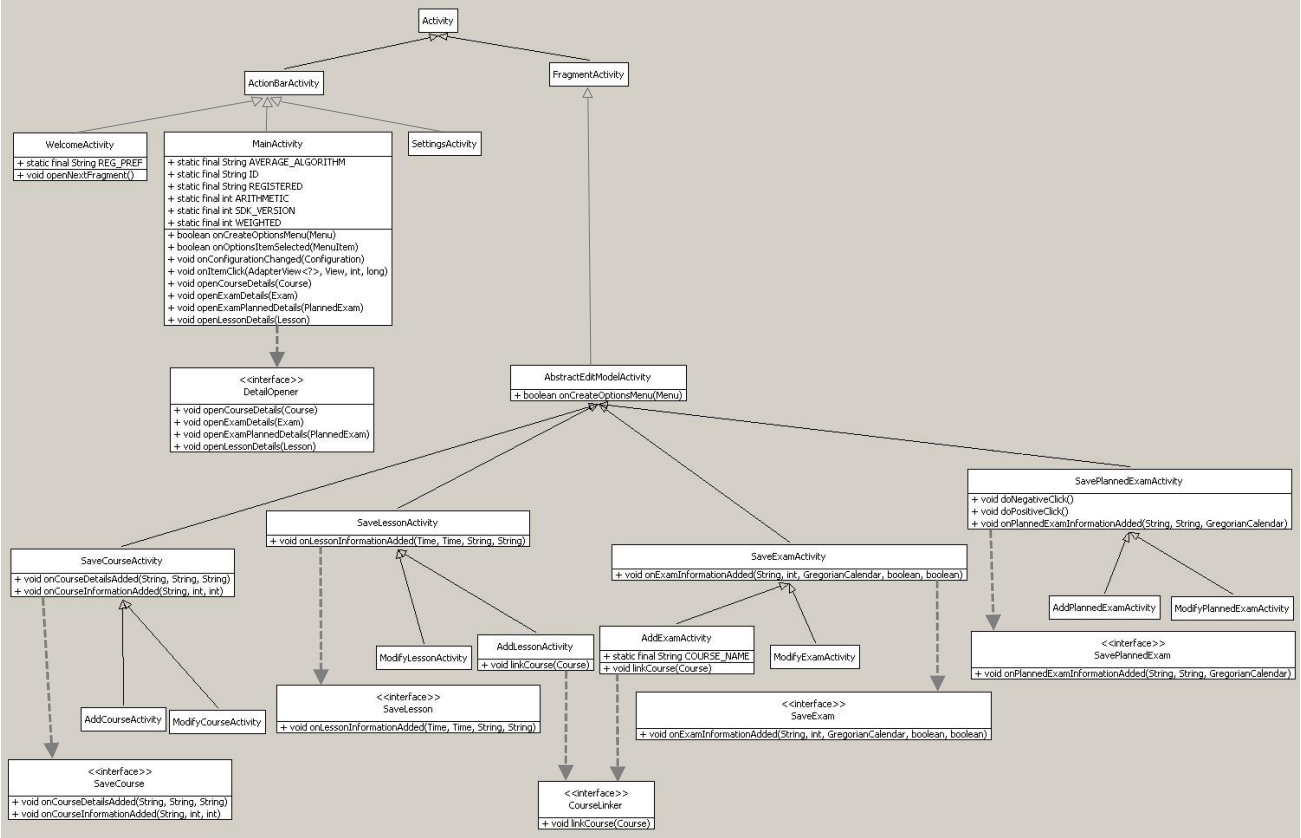
Nelle pagine seguenti andremo a illustrare più nel dettaglio le specifiche di implementazione grazie all'uso di appositi diagrammi UML.



Il diagramma sopra rappresentato descrive la struttura delle classi relative al modello:

- **Model:** è la classe padre di ogni modello. Si tratta di una classe astratta messa a disposizione da ActiveAndroid affinché si possa associare un oggetto Java a una tabella del database.

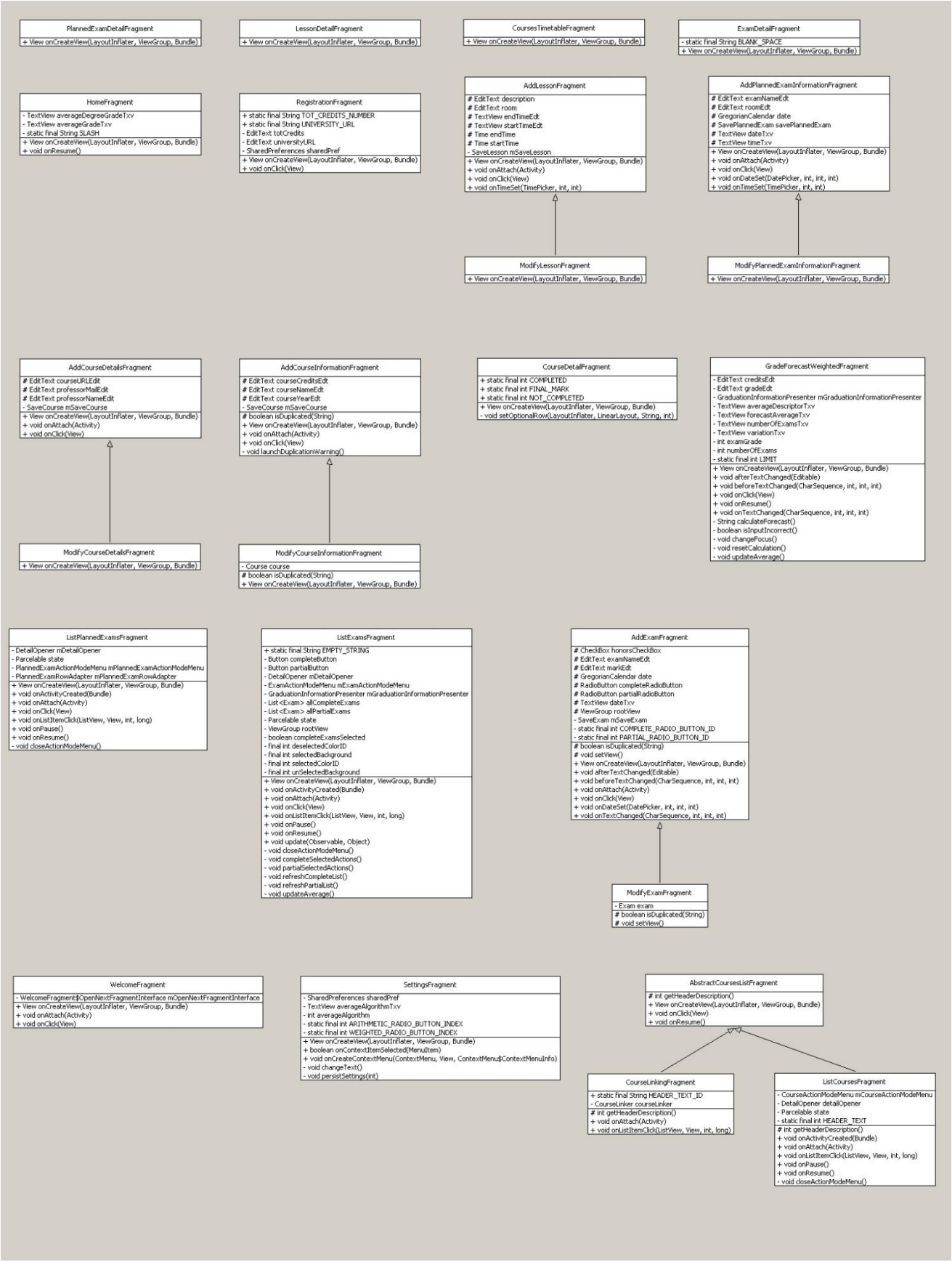
- **Course:** si tratta della classe che gestisce i corsi.
- **Exam:** per ogni corso esistono diversi esami. Il compito di questa classe è gestirli, facendo differenza tra parziali e totali.
- **PlannedExam:** è la classe utilizzata per modellare la pianificazione di esami futuri.
- **Lesson:** per ogni corso esistono diverse lezioni. Il compito di questa classe è gestirne la periodicità, e per farlo abbiamo introdotto le classi **Time** e **Day**.



Il diagramma sopra rappresentato descrive l'architettura delle classi relative alla parte di interazione, che in ambiente Android sono implementate come Activity, con la seguente struttura gerarchica:

- **Activity:** è la classe Activity base fornita da Android, noi ne abbiamo utilizzato due specializzazioni, sempre fornite da Android, **ActionBarActivity** e **FragmentActivity.**
- **WelcomeActivity:** sottoclasse di ActionBarActivity, è un'activity che viene lanciata al primo utilizzo dell'applicazione presentare le proprie funzionalità all'utente e per gestirne la registrazione.
- **SettingsActivity:** sottoclasse di ActionBarActivity, è l'activity che gestisce la visualizzazione e la modifica delle impostazioni
- **MainActivity:** estende ActionBarActivity ed è l'activity principale. Permette di navigare tra le varie sezioni e funzionalità dell'applicazione. Implementa l'interfaccia DetailOpener, che permette ad alcune cassi della view che la utilizzano di comunicare proprio con l'activity.
- **AbstractEditModelActivity:** è una classe astratta, padre per tutte le activity che vanno a modificare il database (inserendo o modificando dati). Estende la classe FragmentActivity, poiché usa consistentemente i Fragment per gestire l'interfaccia grafica.

- **Save(model_name)Actvity:** Come visto precedentemente l'applicazione gestisce principalmente quattro tipi di dato: Course, Exam, Lesson, PLannedExam. Il salvataggio di ognuno di questi è gestito da classi diverse, ma con la stessa struttura. Per ognuno, in particolare, vi è una rispettiva classe astratta, il cui nome è nella forma Save(model_name)Activity. Questa classe definisce le operazioni di base per aiutare un utente a memorizzare il rispettivo dato nel database. A sua volta essa è estesa da due specifiche activity **Add(model_name)Activity** e **Modify(model_name)Activity**, che implementano nello specifico tali operazioni a seconda che l'utente voglia aggiungere o modificare un dato nel database. Ognuna inoltre implementa una specifica interfaccia **Save(model_name),** che permette alle classi di view che la utilizzano di comunicare con l'activity.
- **CourseLinker:** è un'interfaccia implementata dalle classi AddLessonActivity e AddExamActivity, che permette ai fragment che la utilizzano di passare il corso selezionato dall'utente che dovrà essere collegato al dato, di tipo Exam o Lesson, da aggiungere al database.



Il diagramma sopra rappresentato descrive l'architettura delle classi relative alla parte di presentazione, che abbiamo implementato utilizzando i Fragment. Ogni Fragment si occupa della gestione dell'interfaccia grafica relativa ad una delle funzioni offerte dall'applicazione. Sono pertanto prevalentemente indipendenti l'uno dall'altro, ma possono avere caratteristiche comuni. Quando

queste erano importanti, abbiamo sfruttato l'ereditarietà per riutilizzare il codice condiviso. È, ad esempio, il caso dei Fragment per aggiungere e modificare uno specifico dato nel database, che svolgono una funzione quasi uguale e si differenziano per poche righe di codice. Abbiamo per questo deciso di implementare uno dei due estendendo l'altro e aggiungendo le poche operazioni aggiuntive necessarie. Anche la classe **AbstractCourseListFragment** è stata creata con lo scopo di sfruttare il codice comune delle classi figlie. Entrambi i Fragment infatti mostrano all'utente la lista dei corsi salvati, ma per scopi diversi. Con la classe astratta sopra citata abbiamo riunito il codice condiviso in una sola classe, rendendo poi facile estendere la classe per specializzarla a seconda dello scopo.

ORGANIZZAZIONE IN PACKAGE

- **com.csz.unb.controller**: contiene i sorgenti che incapsulano il comportamento dei controller relativi a tutti i fragment e i menu dell'applicazione.
- **com.csz.unb.data:** contiene i sorgenti che modellano l'accesso e il mapping al database.
- **com.csz.unb.interfaces:** contiene le interfacce comuni a tutta l'applicazione.
- **com.csz.unb.scheduling:** contiene i sorgenti relativi alla pianificazione e alle notifiche relativi agli esami.
- **com.csz.unb.statistics:** contiene i sorgenti che calcolano le statistiche (come ad esempio la media ponderata e aritmetica).
- **com.csz.unb.view:** contiene i sorgenti che si occupano della presentazione delle informazione e dell'esperienza utente.

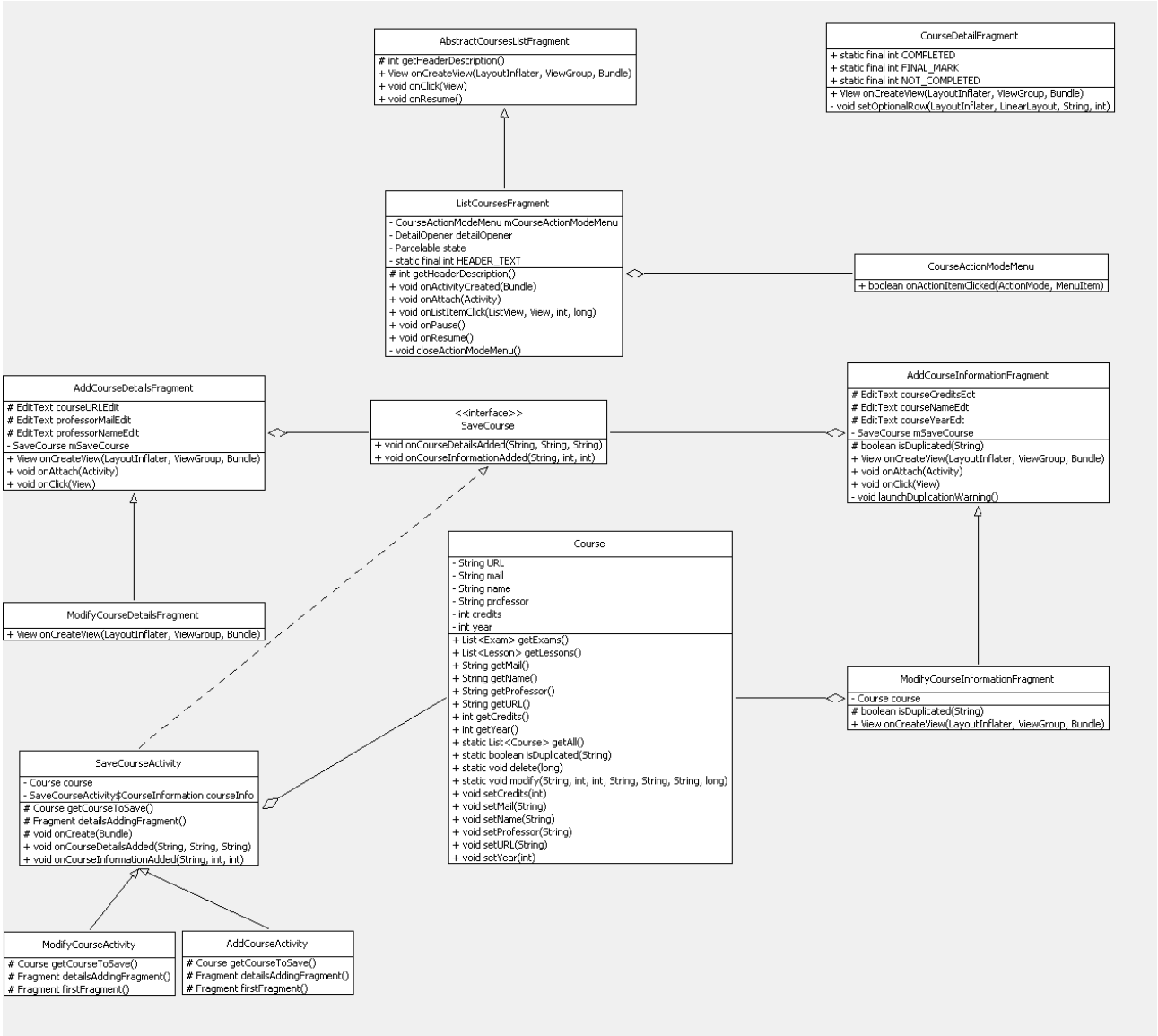
SUDDIVISIONE DEL LAVORO

Nel contesto della realizzazione del progetto il lavoro tra i vari componenti del gruppo è stato suddiviso come segue:

- Campana: Anagrafica corsi, calcolo delle statistiche (e.g. media ponderata, base laurea, etc.), funzione per ipotizzare l'aggiornamento della media
- Scandurra: Promemoria, persistenza dati, orario delle lezioni
- Zamberletti: Gestione esami ed esami parziali, schermata per la registrazione iniziale, hompage

L'analisi del problema è stata svolta collettivamente.

PROGETTAZIONE DI DETTAGLIO: CAMPANA LUCA

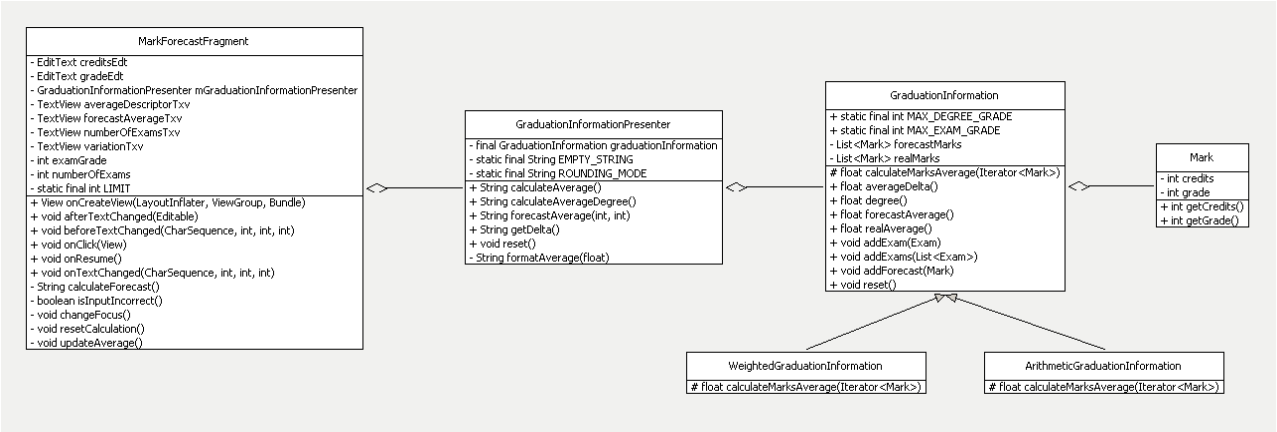


Il diagramma precedente mostra le classi che si occupano della gestione e della presentazione dei corsi all'interno della nostra applicazione. Per quanto riguarda la presentazione dei corsi, abbiamo implementato la classe astratta **AbstractCoursesListFragment**, che possiede le caratteristiche comuni tra tutti i fragment che presentano una lista di corsi, utilizzando la classe concreta **ListCoursesFragment**. Compito di tale fragment è proprio quello di visualizzare tutti i corsi salvati. A tale classe è associato un menu contestuale, **CourseActionModeMenu**, implementazione della classe astratta e generica **AbstractActionModeMenu**, che permette all'utente di eliminare o lanciare l'interfaccia di modifica di un corso. La classe **CourseDetailFragment** permette infine di visualizzare i dettagli di un determinato corso selezionato tra quelli presentati da **ListCoursesFragment**.

La modifica e l'aggiunta di un nuovo corso sono controllate dalla classe astratta **SaveCourseActivity**, più nello specifico implementata rispettivamente da **ModifyCourseActivity** e da **AddCourseActivity**.

L’inserimento dei dati da parte dell’utente è gestito da **AddCourseInformationFragment**, **ModifyCourseInformationFragment** , **AddCourseDetailsFragment** e **ModifyCourseDetailsFragment**, che passano tali dati alla activity **SaveCourseActivity** tramite una interfaccia (**SaveCourse**): l’activity a cui sono associati avrà poi il compito di persisterli nel database. L’utilizzo di interfacce nella comunicazione fragment-activity permette di rendere i fragment facilmente riutilizzabili e associabili ad activity differenti.

Per gestire le funzionalità descritte abbiamo utilizzato il pattern **MVC**, che prevede la suddivisione in presentazione (gestita dai fragment), controllo (affidato alle activity) e modello (qui rappresentato dalla classe Course, che estende la classe astratta Model, fornita dalla libreria ActiveAndroid) e il pattern **Template Method**, tra le classi SaveCourseActivity, AddCourseActivity e ModifyCourseActivity.

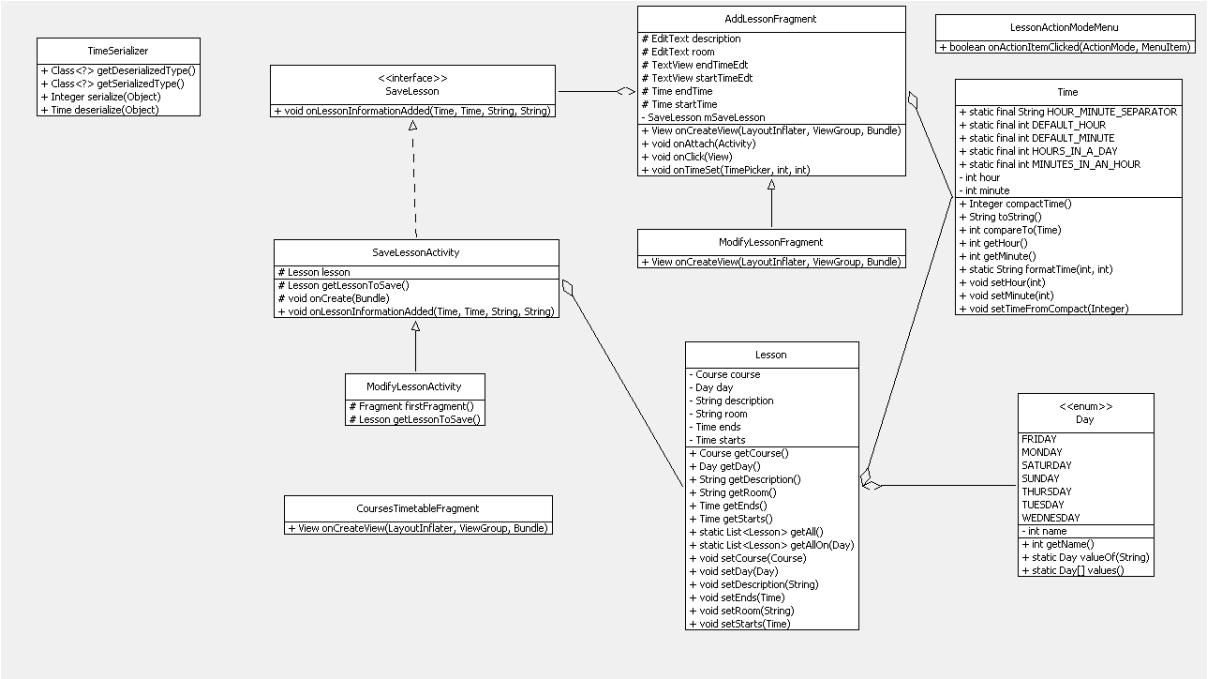


Nel secondo diagramma viene presentata la parte di applicazione che gestisce la previsione della media a fronte dell’inserimento di esami ipotizzati.

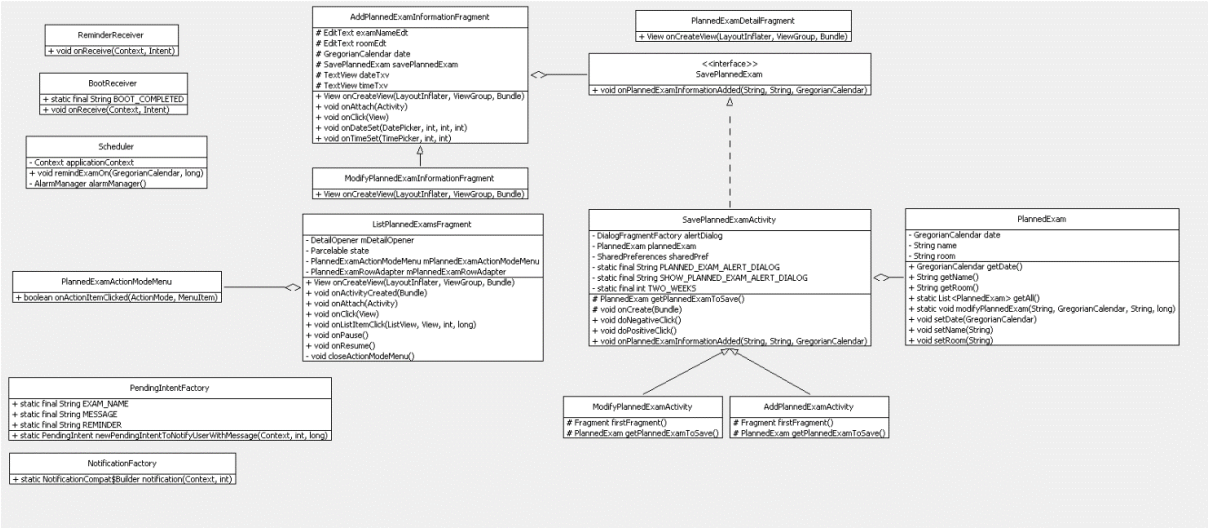
Il fragment **MarkForecastFragment** fornisce l’interfaccia utente per l’inserimento del voto ipotizzato ed eventualmente dei crediti relativi (questi ultimi vengono considerati solo se si è scelto il calcolo della media ponderata dalle impostazioni) e visualizza la nuova media prevista. La parte di calcolo vera e propria è eseguita dalla classe astratta **GraduationInformation**, implementata dalle classi concrete **WeightedGraduationinformation** e **ArithmeticGraduationinformation**, che forniscono i due algoritmi per il calcolo della media ponderata e aritmetica. La classe **Mark** memorizza nei suoi due campi il voto e i crediti per un dato esame. Infine è presente una classe wrapper di GraduationInformation, **GraduationInformationPresenter**, con il compito di convertire in stringa le medie calcolate e di fornirle a MarkForecastFragment.

In questa parte si è fatto uso del pattern **Template Method**, con la classe astratta GraduationInformation che racchiude il codice comune e con le classi concrete WeightedGraduationInformation e ArithmeticGraduationInformation che determinano l’algoritmo di calcolo della media.

PROGETTAZIONE DI DETTAGLIO: SCANDURRA ANTONIO



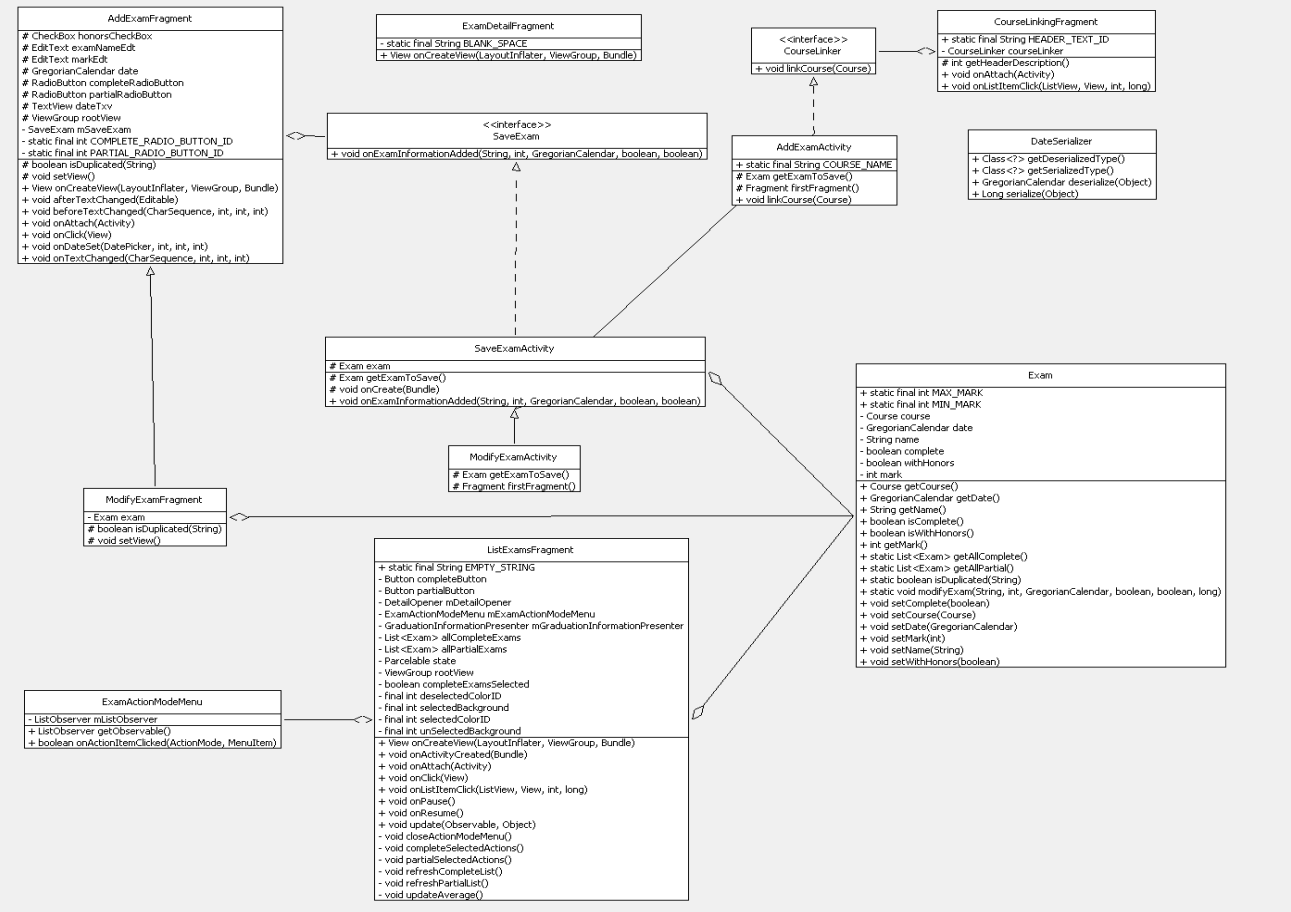
Il diagramma sopra illustrato ruota attorno alla gestione del modello **Lesson**. Prima di evidenziare come tale oggetto viene utilizzato, è interessante notare come si sia fatto uso di due Value Object per rappresentare dei concetti inerenti a questo dominio: **Day** e **Time** (quest'ultimo viene serializzato su database attraverso **TimeSerializer**). Il menu contestuale **LessonActionModelMenu** agisce come entry point per accedere alle varie funzionalità. **SaveLessonActivity** e **ModifyLessonActivity** si occupano di inizializzare i relativi fragment (**AddLessonFragment** e **ModifyLessonFragment**) e di salvare su db (interrogando ActiveAndroid) le informazioni generate da questi ultimi implementando l'interfaccia onLessonInformationAdded. Una possibile scelta implementativa poteva essere quella di adoperare ActiveAndroid direttamente all'interno dei fragment: questi ultimi, però, dovrebbero avere come unico compito quello di orchestrare l'interazione utente; comunicare con il db (o per meglio dire l'ORM), avrebbe aumentato le responsabilità dei fragment rendendo vano, inoltre, l'incapsulamento. Infine, **CoursesTimetableFragment** si occupa di gestire la visualizzazione delle lezioni precedentemente salvate sotto forma di tabellone orario.



Il diagramma sopra illustrato descrive la gestione del modello **PlannedExam**, ovvero la pianificazione degli esami. Prima di evidenziare come tale oggetto viene utilizzato, è interessante notare come si sia gestito lo scheduling delle notifiche. **Scheduler** è una façade che gestisce la gestione delle notifiche in modo trasparente per i client, che devono soltanto passare la data in cui vogliono che una notifica venga visualizzata. Questo oggetto si occupa di orchestrare lo scheduling così come definito da Android e di richiamare la factory **PendingIntentFactory**, che restituisce un oggetto che attiverà **ReminderReceiver** nell'esatto momento in cui la notifica dovrà essere mostrata. Quest'ultimo, attraverso **NotificationFactory**, crea una Android Notification, e la mostra all'utente. È stato fatto uso, inoltre, di **BootReceiver** per poter rischedulare tutte le notifiche che verrebbero altrimenti perse al riavvio del dispositivo. Il menu contestuale **PlannedExamActionModelMenu** agisce come entry point per accedere alle varie funzionalità. **SavePlannedExamActivity** è un'abstract class implementata da **ModifyPlannedExamActivity** e **AddPlannedExamActivity**, che si occupa di inizializzare i relativi fragment (**AddPlannedExamInformationFragment** e **ModifyPlannedExamInformationFragment**) e di salvare su db (interrogando ActiveAndroid) le informazioni generate da questi ultimi implementando l'interfaccia onPlannedExamInformationAdded. Come prima, abbiamo scelto di interrogare ActiveAndroid in questo layer per non spezzare l'incapsulamento. Infine, **ListPlannedExamsFragment** si occupa di gestire la visualizzazione degli esami programmati.

PROGETTAZIONE DI DETTAGLIO: ZAMBERLETTI ANDREA

Il seguente diagramma mostra la struttura delle classi relative alla parte, sviluppata da Zamberletti Andrea, per la gestione degli esami:



Tale parte è incentrata sulla gestione del modello **Exam**, rappresentante un esame.
In particolare, la classe **SaveExamActivity** si occupa del controllo per le funzioni di salvataggio di istanze di Exam nel database. Sebbene infatti la persistenza dei dati sia gestita dalla libreria esterna ActiveAndroid, l'activity deve comunicare con le classi relative all'interfaccia grafica per reperire le informazioni inserite dall'utente. Questo viene fatto attraverso l'interfaccia **SaveExam**, implementata da SaveExamActivity ed utilizzata da **AddExamFragment** e dalla sua sottoclasse **ModifyExamFragment**, le classi che si occupano proprio di gestire l'inserimento dei dati da parte dell'utente, per comunicare l'avvenuta aggiunta e passare le informazioni ricevute all'activity stessa.
La classe SaveExamActivity, inoltre, è specializzata da due sottoclassi: **AddExamActivity** e **ModifyExamActivity**, a seconda che l'esame da salvare sia nuovo o già presente nel database. La prima di queste due classi si deve addizionalmente occupare di associare il nuovo esame ad uno dei corsi presenti nel database. Questo viene fatto tramite l'interfaccia **CourseLinker**, implementata da AddExamActivity ed utilizzata da **CourseLinkingFragment**, classe che si occupa di mostrare una lista dei corsi salvati, per passargli il corso una volta che è stato selezionato dall'utente. Una lista di tutti gli esami memorizzati è invece visualizzabile grazie alla classe **ListExamsFragment**; questa gestisce anche un menu contestuale, implementato all'interno della classe **ExamActionModeMenu** che permette all'utente di eliminare o accedere alla sezione per la modifica di un esame. Sempre da ListExamsFragment, l'utente può infine accedere ad un'altra schermata gestita dalla classe **ExamDetailFragment**, per la visualizzazione delle informazioni relative ad un esame.

DateSerializer, infine, è una classe utilizzata da ActiveAndroid per permettergli di memorizzare la data dell'esame, espressa attraverso il tipo non primitivo GregorianCalendar.

Oltre al pattern architetturale **MVC**, sono stati utilizzati altri due pattern nello sviluppo di questa parte: il pattern **Template Method**, sfruttato all'interno della classe astratta SaveExamActivity, e il pattern **Observer** usato da ListExamFragment per tenere d'occhio l'eliminazione e la modifica di esami, aggiornando la media di conseguenza.

NOTE FINALI

Il processo di sviluppo è stato spirale, caratterizzato da operazioni di refactoring e cambiamenti della progettazione in corso d'opera. Il lavoro è stato svolto in parte in gruppo e in parte individualmente in modo parallelo, facendo uso degli strumenti forniti da Mercurial.

Le difficoltà maggiori sono sorte dalle scarse conoscenze sul sistema Android da parte di tutti i componenti del gruppo. Parte delle 300 ore totali sono state dunque spese per l'apprendimento della logica di Android, rallentando il processo di sviluppo.