SAPIENZA
UNIVERSITÀ DI ROMA

Piattaforma desktop/mobile per la gestione della domanda e dell'offerta di servizi

Facoltà di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Informatica

Candidato

Alice Guercio

1184160

Relatore

Gianni Orlandi

A/A 2014/2015

RACCOLTA DEI REQUISITI

DESCRIZIONE DELLA REALTA'

Si vuole realizzare una applicazione web e Android per la gestione di offerte di ripetizioni, lezioni private, annunci, mantenuti in un database.

Un annuncio è composto da un titolo, un testo, una materia di riferimento, un docente, la città e l'indirizzo della sede in cui è erogata la lezione, un prezzo, un identificativo e uno o più livelli, relativi al grado di istruzione.
Ad ogni annuncio possono essere associati dei giudizi, espressi dagli utenti. Un giudizio è costituito da un autore, una data, un rating ed un eventuale commento testuale.
Ogni utente può commentare una ed una sola volta il singolo annuncio, onde evitare il proliferare di giudizi parziali. Potrà in ogni caso modificare il suo giudizio in qualsiasi momento, qualora la qualità dell'insegnamento subisse cambiamenti (miglioramenti o peggioramenti nel tempo).
Ogni materia è caratterizzata da un nome identificativo e una categoria di appartenenza.
Un utente è caratterizzato da un identificativo, un nome, un cognome, una password e una mail e può pubblicare giudizi.
Un insegnante è una specializzazione di utente, che può pubblicare annunci. E' caratterizzato da un telefono, una città e un eventuale titolo di studi.
Un admin è una specializzazione di utente ed è in grado di eliminare qualsiasi giudizio.

DESCRIZIONE DETTAGLIATA

L'applicazione gestisce dati relativi ad annunci, utenti, insegnanti, giudizi, materie.

Tipologia di utenti:
- Utente: caratterizzato da identificativo, nome, cognome, username, email, password
- Insegnante: caratterizzato da identificativo, nome, cognome, username, titolo di studi, città, telefono
- Admin: caratterizzato da identificativo, nome, cognome, username, email, password

L'utente può consultare gli annunci, effettuare ricerche tramite filtri, esprimere giudizi e commenti, registrarsi come insegnante, modificare i propri dati.

L'insegnante può consultare gli annunci, effettuare ricerche, esprimere giudizi sugli annunci che non lo coinvolgono, modificare o eliminare i propri annunci, modificare i propri dati. All'atto della composizione dell'annuncio può selezionare la materia dall'elenco fornito dal sistema, ma non può indicarne di nuove. Questa limitazione dei poteri dell'utente insegnante è motivata dal timore che l'utente possa rendere il sistema disordinato, introducendo ad esempio doppioni.

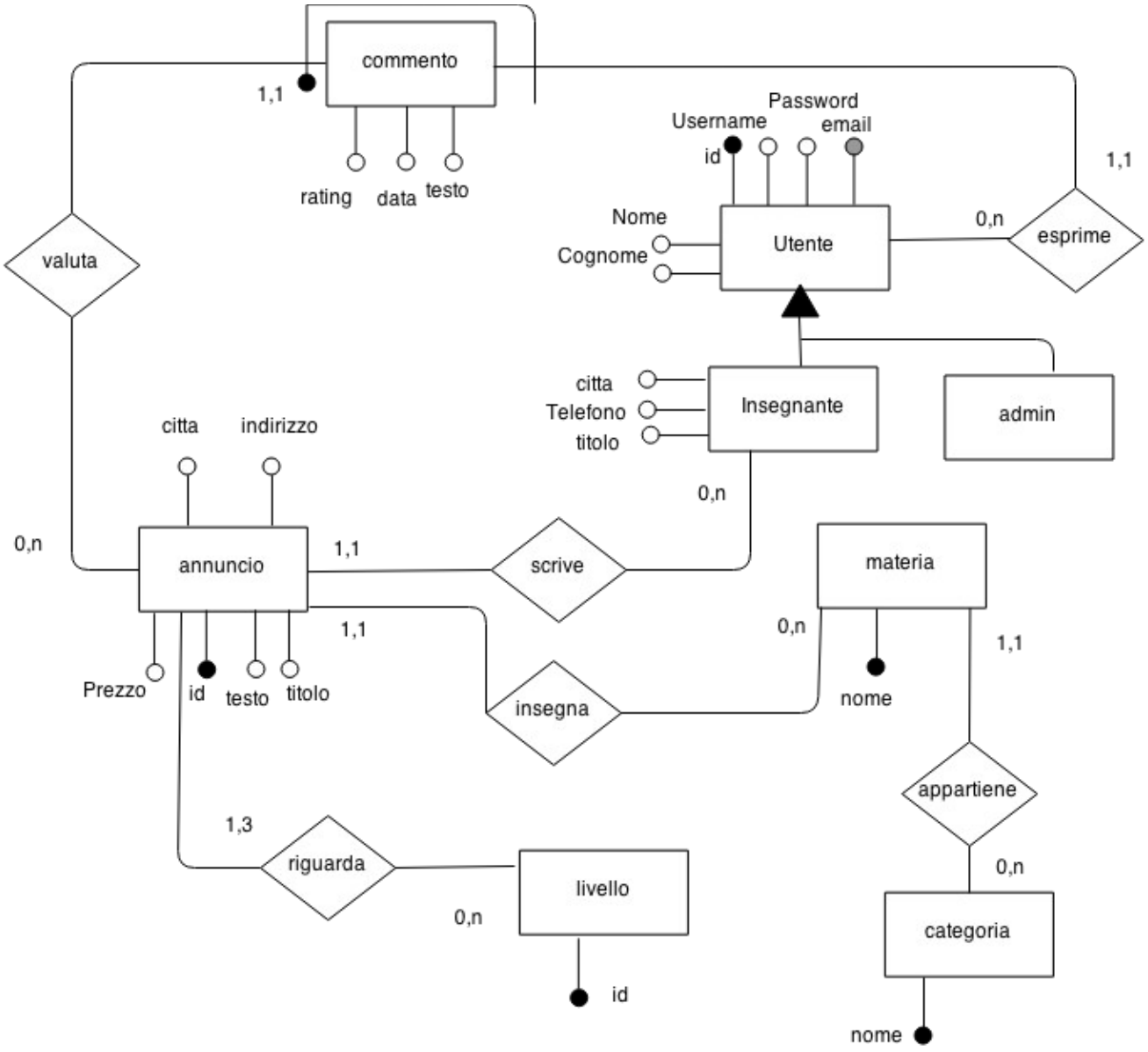
L'admin può eliminare i commenti di qualsiasi utente, visualizzare gli annunci, effettuare ricerche, esprimere giudizi e commenti, registrarsi come insegnante, modificare i propri dati, visualizzare l'elenco utenti e i loro dettagli.

L’annuncio è caratterizzato da : identificativo, insegnante, titolo, testo, materia, città, indirizzo e uno o più livelli (ELEMENTARE, SUPERIORE, UNIVERSITARIO), che esprimono il grado di istruzione del corso.

Ad ogni annuncio possono associarsi dei giudizi, caratterizzati da: testo, data, autore, annuncio relativo e rating, espresso da un valore numerico da 1 a 5. Autore e annuncio sono gli identificativi del giudizio, poiché non è ammesso che il singolo utente possa esprimere più di un giudizio per annuncio. L’utente può modificare o eliminare i giudizi che ha scritto.

PROGETTAZIONE DELLA BASE DATI

SCHEMA ER



VINCOLI NON ESPRIMIBILE A LIVELLO ER

- L’insegnante che ha scritto l’annuncio, non può esprimere una valutazione su di esso.
-

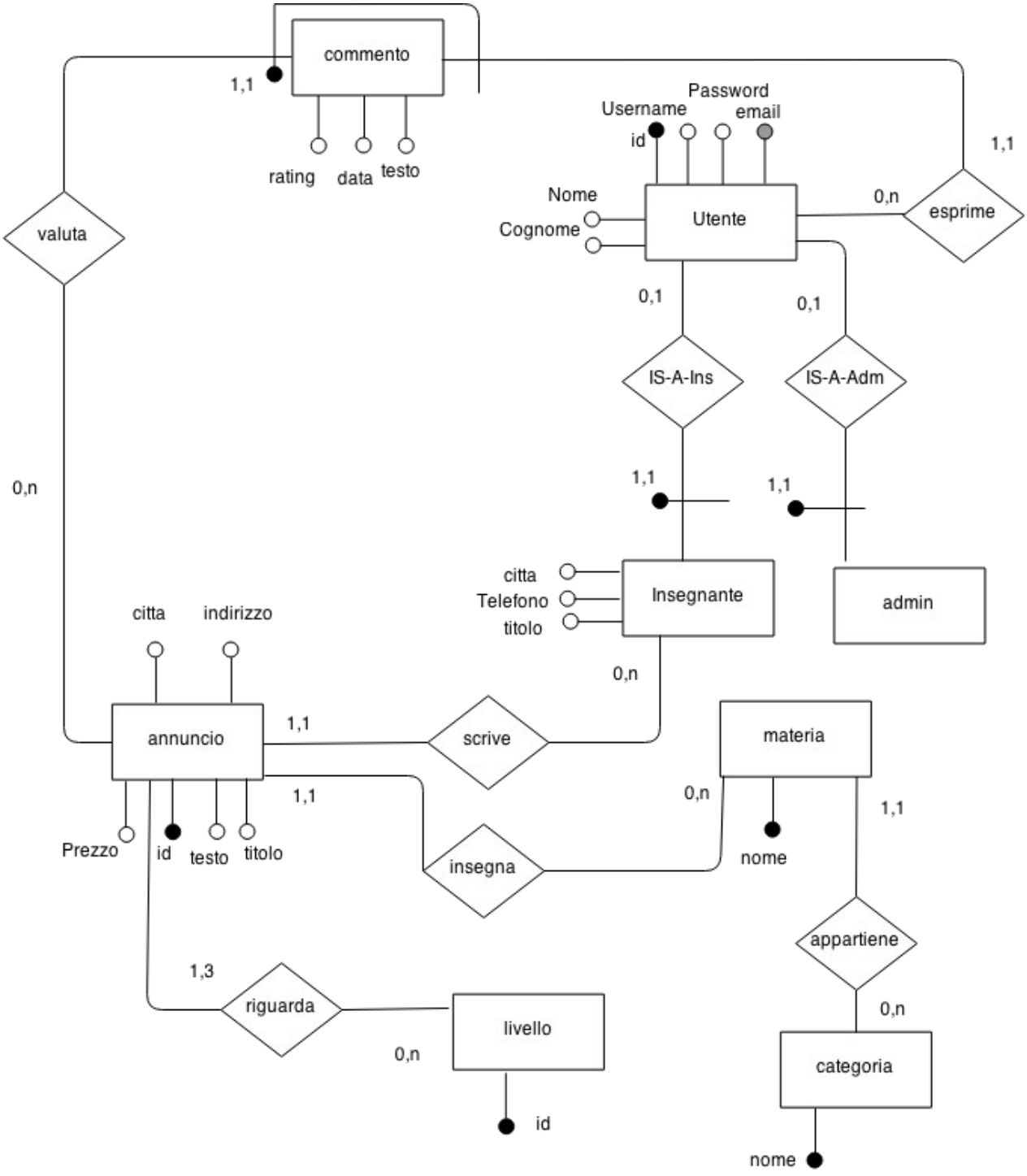
GLOSSARIO DELLE ENTITA'

Entità	Descrizione	Attributi	Identificatore
Utente	utente dell'applicazione	username nome cognome password mail id	id
Insegnante	tipo di utente che può pubblicare annunci	id città telefono titolo	id
Materia	La materia offerta nell'annuncio	nome	nome
Categoria	Definisce l'area relativa alla materia	nome	nome
Annuncio	Offerta di lezione privata da parte di un insegnante	Prezzo id titolo testo città indirizzo	id
Livello	Esprime il grado di insegnamento (elementare, superiore universitario)	id	id
Commento	Esprime un giudizio da parte dell'utente sull'annuncio	rating data testo	annuncio utente
Utente	L'utente dell'applicazione	id username password nome cognome email	id
Insegnante	L'utente che pubblica annunci	telefono città titolo	id
Admin	L'amministratore	id	id

GLOSSARIO DELLE RELAZIONI

Relazione	Descrizione	Entità coinvolte	Attributi
Esprime	relazione che associa l'utente al commento	utente commento	
Valuta	Associazione che esprime la valutazione di un annuncio	commento annuncio	
Scrive	associazione che indica l'insegnante, autore dell'annuncio	annuncio insegnante	
Insegna	associazione che indica la materia di interesse dell'annuncio	annuncio materia	
Riguarda	associa i livelli all'annuncio	livello annuncio	
Appartiene	indica la categoria di appartenenza della materia	materia categoria	

SCHEMA ER RISTRUTTURATO



Si esegue la ristrutturazione per semplificare la successiva fase di traduzione, eliminando quei costrutti non direttamente traducibili e per tenere conto dell'efficienza

Attività della ristrutturazione dello schema e-r:
- Analisi delle ridondanze.
- Eliminazione degli attributi multi-valore
- Eliminazione delle ISA e generalizzazioni
- Specifica di ulteriori vincoli esterni

Le uniche attività di ristrutturazione compiute sono state: l'eliminazione delle ISA.
Non consideriamo l'attributo città, presente nell'entità Insegnante e nell'entità Annuncio, come una ridondanza. Poiché l'insegnante potrebbe tenere una lezione in una città differente da quella di appartenenza.

SCHEMA LOGICO

Per ottimizzare gli accessi al database si è deciso di operare alcuni accorpamenti.

Utente (id, username, password, email, nome, cognome)
chiave: email
Insegnante (id, città, telefono, titolo*)
Foreign key: Insegnante[id] C Utente [id]
Admin (id)
Foreign key: Admin [id] C Utente [id]
Materia (nome, categoria)
Annuncio (id, titolo, testo, materia, prezzo, città, indirizzo, insegnante)
Foreign key: Annuncio [materia] C Materia [nome]
Foreign key: Annuncio [insegnante] C Insegnante [id]
Livello (id)
Riguarda (annuncio, livello)
Foreign key: Riguarda [annuncio] C Annuncio [id]
Foreign key: Riguarda [livello] C Livello [id]
Commento (data, rating, testo, annuncio, utente)
Foreing key: Commento [annuncio] C Annuncio [id]
Foreign key: Commento [utente] C Utente [id]

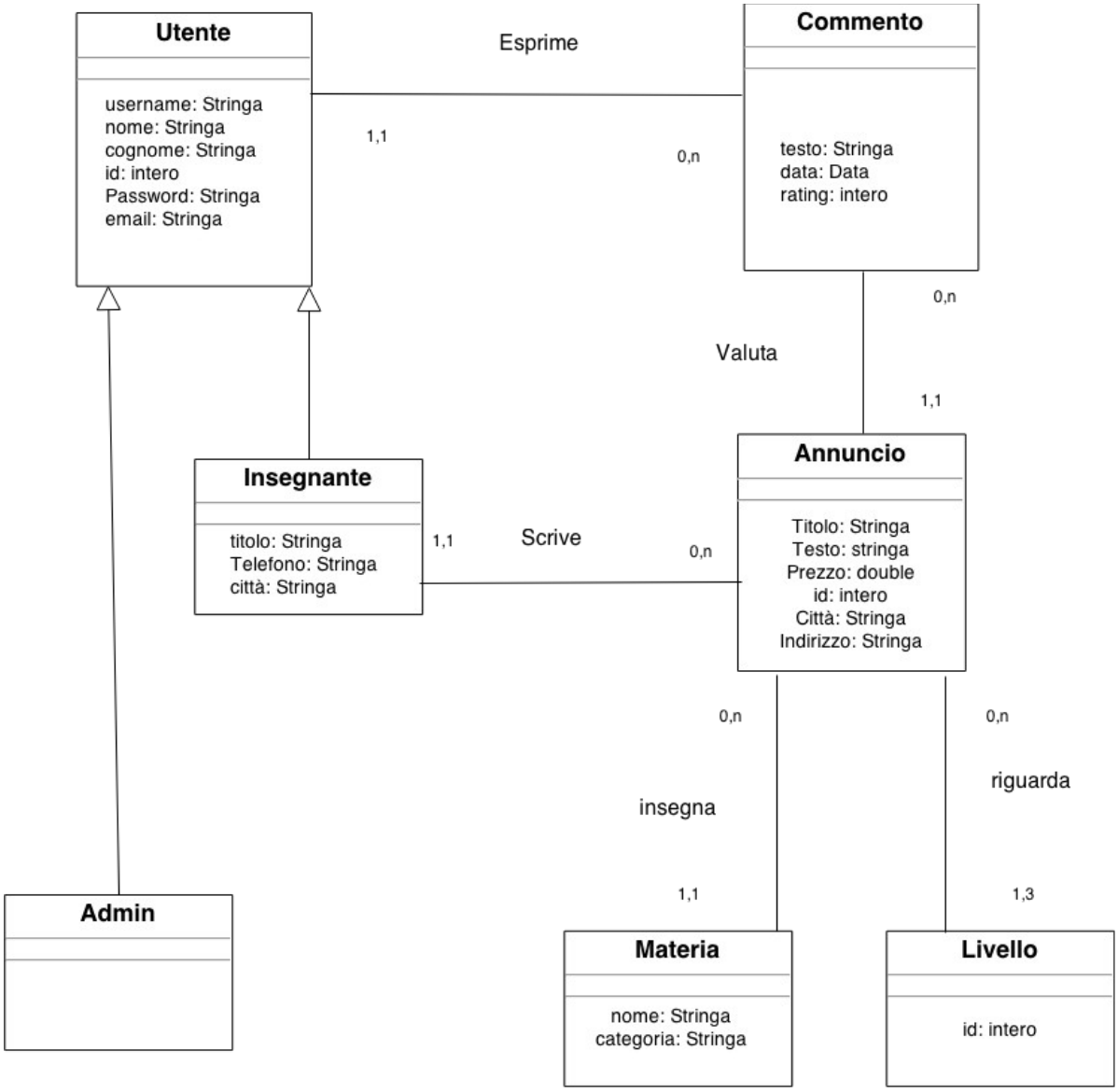
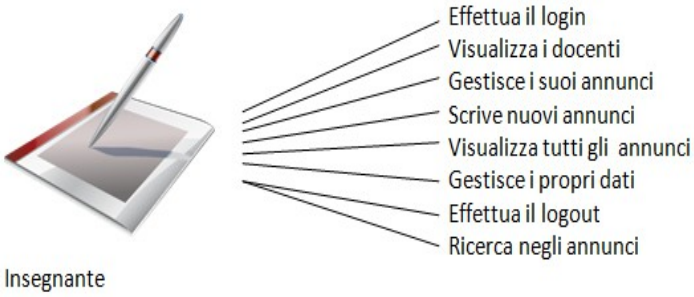


DIAGRAMMA DELLE CLASSI UML

USE CASE DIAGRAM



Insegnante

Utente

Admin

Descrizione delle funzionalità in termini del diagramma delle classi

Nome Funzione	Login
Descrizione	Consente l’accesso al sistema
Attori coinvolti	Utente, Admin, Insegante
Algoritmo	confronta i dati inseriti con quelli registrati sul database e crea una sessione

Nome Funzione	Registrazione classica
Descrizione	Consente di registrare i propri dati di accesso sul database
Attori coinvolti	Utente
Algoritmo	inserisci dati nel database

Nome Funzione	Registrazione Insegnante
Descrizione	Consente di registrare i propri dati di accesso sul database
Attori coinvolti	Insegnante
Algoritmo	inserisci dati nel database

Nome Funzione	Modifica dati account
Descrizione	Consente di cambiare i dati del proprio account
Attori coinvolti	Utente, Admin, Insegnante
Algoritmo	Modifica i dati nel database

Nome Funzione	Modifica Annuncio
Descrizione	Consente di cambiare i dati relativi ad un annuncio
Attori coinvolti	Insegnante
Algoritmo	Modifica i dati nel database

Nome Funzione	Modifica Commento
Descrizione	Consente di cambiare i dati relativi ad un commento
Attori coinvolti	Utente, Admin, Insegnante
Algoritmo	Modifica i dati nel database

Nome Funzione	Elimina annuncio
Descrizione	Consente di cancellare il proprio annuncio
Attori coinvolti	Insegnante
Algoritmo	Elimina i dati nel database degli annunci

Nome Funzione	Visualizza Annunci
Descrizione	Visualizza i dati contenuti nei database degli annunci
Attori coinvolti	Utente, insegnante, admin
Algoritmo	Carica i dati dal database di annunci

Nome Funzione	Visualizza Docenti
Descrizione	Visualizza i dati contenuti nei database dei docenti
Attori coinvolti	Utente, insegnante, admin
Algoritmo	Carica i dati dal database di docenti

Nome Funzione	Ricerca annunci
Descrizione	Visualizza i dati contenuti nel database annunci in base ai criteri di ricerca scelti
Attori coinvolti	Utente, insegnante, admin
Algoritmo	Carica i dati dai database di annuncio in base ai filtri di ricerca selezionati

Nome Funzione	Inserisci annuncio
Descrizione	Inserisce i dati inviati dall'insegnante nel database annuncio
Attori coinvolti	Insegnante
Algoritmo	Inserisce i dati nel database di annuncio

Nome Funzione	Inserisci commento
Descrizione	Inserisce i dati inviati inviati dall'utente nel database di commenti
Attori coinvolti	Utente, insegnante, admin
Algoritmo	Inserisce i dati nel database di commento

Nome Funzione	Elimina commento
Descrizione	Elimina dal database di commenti
Attori coinvolti	Utente, admin
Algoritmo	elimina i dati nel database di commento

Responsabilità sulle associazioni

Prima di eseguire effettivamente la realizzazione, bisogna considerare la responsabilità delle associazioni. Diciamo che una classe C ha responsabilità sull'associazione A, quando per ogni oggetto x che è istanza di C, vogliamo poter eseguire opportune operazioni sulle istanze di A a cui x partecipa.
I criteri per decidere la responsabilità sulle associazioni sono i seguenti:
1) Si evince dai requisiti che per ogni oggetto x, istanza di C, vogliamo poter eseguire almeno una delle operazioni di aggiunta, cancellazione, aggiornamento o inserimento.
2) Esiste una operazione nello use case che determini una responsabilità
3) Vincoli di molteplicità implicano responsabilità

ASSOCIAZIONE	CLASSI	RESPONSABILITA’
Esprime	utente commento	$SI_{2,3}$ $SI_{1,2}$
Valuta	commento annuncio	$SI_{1,2}$ $SI_{1,2,3}$
Scrive	insegnante annuncio	$SI_{1,2,3}$ $SI_{1,2}$
Riguarda	annuncio livello	$SI_{1,2}$ $SI_{1,2,3}$
Insegna	annuncio materia	$SI_{1,2}$ $SI_{1,2,3}$

PROGETTAZIONE FISICA DELLA BASE DI DATI

Il Resource Management Layer dell’applicazione è stato realizzato usando il Database Management System relazionale MySql.
Si è scelto l'engine InnoDB per supportare i vincoli di foreign key

CODICE DELLA BASE DATI

```
CREATE DATABASE IF NOT EXISTS `u561888553_tesi`;

--
--    Struttura della tabella `admin`
--
CREATE TABLE IF NOT EXISTS `admin` (
        `id` int (31) NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

----------------------------------------
--
--    Struttura della tabella `annuncio`
--
CREATE TABLE IF NOT EXISTS `annuncio` (
        `insegnante` int (31) NOT NULL,
        `prezzo` double NOT NULL,
        `id` int(31) NOT NULL AUTO_INCREMENT,
        `testo` text COLLATE utf8_unicode_ci NOT NULL,
        `titolo` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
        `materia` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
        `citta` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
        `indirizzo` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
```

```
        PRIMARY KEY (`id`),
        KEY `annuncio1_fk` (`insegnante`),
        KEY `annuncio2_fk` (`materia`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=28 ;

--------------------------------------



--
-- Struttura della tabella `commento`
--

CREATE TABLE IF NOT EXISTS `commento` (
        `testo` text COLLATE utf8_unicode_ci,
         `data` date NOT NULL,
         `rating` int(10) NOT NULL,
         `utente` int(31) NOT NULL,
         `annuncio` int(31) NOT NULL,
         PRIMARY KEY (`utente`,`annuncio`),
         KEY `commento_fk_annuncio` (`annuncio`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------



--
-- Struttura della tabella `insegnante`
--

CREATE TABLE IF NOT EXISTS `insegnante` (
         `id` int(31) NOT NULL,
         `titolo` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
         `citta` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
         `telefono` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
         PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------



--
-- Struttura della tabella `livelloAnnunci`
--

CREATE TABLE IF NOT EXISTS `livelloAnnunci` (
        `annuncio` int(31) NOT NULL,
          `livello` int(10) NOT NULL,
          PRIMARY KEY (`annuncio`,`livello`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------



--
```

```
-- Struttura della tabella `materia`
--

CREATE TABLE IF NOT EXISTS `materia` (
          `nome` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
          `categoria` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
           PRIMARY KEY (`nome`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

--
-- Dump dei dati per la tabella `materia`
--

INSERT INTO `materia` (`nome`, `categoria`) VALUES
('altro', 'altro'),
('analisi', 'matematica'),
('arte', 'artistico'),
('biologia', 'scientifico'),
('cake design', 'cucina'),
('canto', 'musica'),
('chimica', 'scientifico'),
('chitarra', 'musica'),
('cinese', 'lingue'),
('danza', 'danza'),
('filosofia', 'umanistico'),
('fisica', 'scientifico'),
('francese', 'lingue'),
('giapponese', 'lingue'),
('greco', 'umanistico'),
('informatica', 'informatica'),
('inglese', 'lingue'),
('italiano', 'umanistico'),
('latino', 'umanistico'),
('lingue straniere', 'lingue'),
('matematica', 'matematica'),
('musica', 'musica'),
('pianoforte', 'musica'),
('spagnolo', 'lingue'),
('tedesco', 'lingue');

-- --------------------------------------------------------

--
-- Struttura della tabella `utente`
--

CREATE TABLE IF NOT EXISTS `utente` (
         `id` int(31) NOT NULL AUTO_INCREMENT,
          `username` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
           `nome` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
          `cognome` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
          `password` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
          `email` varchar(31) COLLATE utf8_unicode_ci NOT NULL,
          PRIMARY KEY (`id`),
```

```
        UNIQUE KEY `username` (`username`,`email`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=10 ;

--
-- Dump dei dati per la tabella `utente`
--

INSERT INTO `utente` (`id`, `username`, `nome`, `cognome`, `password`, `email`) VALUES
(1, 'holyna', 'Alice', 'Guercio', '$1$x7EO/lwE$pKADEUBKPdd/65ld1dI5D/', 'kimma@hotmail.com');


--
-- Limiti per le tabelle
--

--
-- Limiti per la tabella `admin`
--
ALTER TABLE `admin`
  ADD CONSTRAINT `admin_fk` FOREIGN KEY (`id`) REFERENCES `utente` (`id`) ON UPDATE
CASCADE;

--
-- Limiti per la tabella `annuncio`
--
ALTER TABLE `annuncio`
  ADD CONSTRAINT `annuncio2_fk` FOREIGN KEY (`materia`) REFERENCES `materia` (`nome`) ON
DELETE CASCADE,
  ADD CONSTRAINT `annuncio1_fk` FOREIGN KEY (`insegnante`) REFERENCES `insegnante` (`id`)
ON DELETE CASCADE;

--
-- Limiti per la tabella `commento`
--
ALTER TABLE `commento`
  ADD CONSTRAINT `commento_fk_annuncio` FOREIGN KEY (`annuncio`) REFERENCES `annuncio`
(`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `commento_fk_utente` FOREIGN KEY (`utente`) REFERENCES `utente` (`id`);

--
-- Limiti per la tabella `insegnante`
--
ALTER TABLE `insegnante`
  ADD CONSTRAINT `insegnante_fk` FOREIGN KEY (`id`) REFERENCES `utente` (`id`) ON UPDATE
CASCADE;

--
-- Limiti per la tabella `livelloAnnunci`
--
ALTER TABLE `livelloAnnunci`
  ADD CONSTRAINT `livelloAnnunci_fk` FOREIGN KEY (`annuncio`) REFERENCES `annuncio` (`id`)
ON DELETE CASCADE;
```

POLITICHE ADOTTATE

Per rappresentare i livelli scolastici (elementare, superiore e universitario) si sarebbe potuto usare il tipo ENUM, ma si è preferito optare per l'uso di costanti intere. In questo modo non sarà necessario ristrutturare il database, qualora si decidesse di aggiungere nuovi livelli.

Si è conferito all'Admin il potere di cancellare i commenti degli utenti, per garantire la possibilità di rimuovere eventuali giudizi iniqui o inappropriati.

Object relational mapping

A causa dell'impendance mismatch, cioè la mancanza di corrispondenza tra il modello relazionale della base di dati e l'approccio orientato agli oggetti dell'applicazione, si è scelto di impiegare un meccanismo di Object-relational Mapping (ORM) per mappare coerentemente gli oggetti di tipo persistente dell'applicazione sul database.

Per una maggiore efficienza si è scelto di impiegare un approccio a DAO (Data Access Objects) di tipo non "puro". Il meccanismo DAO prevede uno strato dell'applicazione che si occupa di gestire la comunicazione col database; specifiche classi DAO, ciascuna rappresentate una classe di dominio, vengono usate per le operazioni di lettura, creazione, modifica, cancellazione delle classi, mentre per le operazioni che coinvolgono più classi, si impiegano le classi DCS (Data Control Services), che raccogliendo la logica di classi diverse in una sola classe, permettono di snellire il codice, aumentando la coesione e diminuendo l'accoppiamento tra classi.

Ogni classe di dominio mette a disposizione metodi di inserimento, caricamento, update ed eventualmente cancellazione, al cui interno richiameranno i metodi della classe DAO.

In questo modo, classi di dominio e controller sono perfettamente indipendenti dalla base di dati.

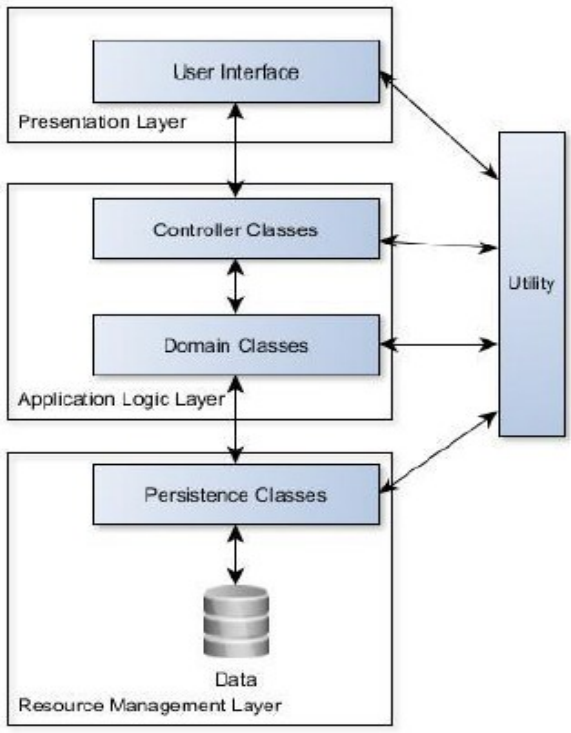
Gestione delle eccezioni: catturerò le eccezioni a livello di DAO e DCS, incapsulandole in una classe eccezione, definita appositamente per l'applicazione (reperibile nella cartella delle utility).

Per la comunicazione col database del server è stata utilizzata l'estensione PDO (PHP Data Objects). Per far sì che venga creato, per ogni sessione, un solo oggetto PDO per tutte le connessioni, si è impiegata la soluzione con classe Singleton. Il singleton è un design pattern, usato per assicurare che una classe abbia una sola istanza ed un unico punto di accesso globale. Un metodo statico si carica della

responsabilità di assicurare che nessuna altra istanza venga creata oltre la prima, restituendo contemporaneamente un riferimento all'unica esistente.

```
class PDOFactory {

	public static function GetPDO ($strDSN, $strUser, $strPass, $arPar) {
		$strKey=md5(serialize (array ($strDSN, $strUser, $strPass, $arPar)));
		if (!(@$GLOBALS ['PDOS'] [$strKey] instanceof PDO))
			$GLOBALS ['PDOS'] [$strKey] = new PDO ($strDSN, $strUser, $strPass,
$arPar);
		$GLOBALS ['PDOS'] [$strKey]->setAttribute (PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
		return ($GLOBALS ['PDOS'] [$strKey]);
	}
}
```



Architettura

Per questa applicazione si è scelta una architettura di tipo three-tier, costituita dai tre livelli: Presentation Layer (demandato a presentare le informazioni e a interagire con l'utente), Application Logic Layer (logica della applicazione) e Resource Management Layer (accesso ai dati). I livelli sono completamente disaccoppiati.

Persistence classes: Classi Dao e classi DCS, si occupano di gestire l'accesso ai dati.

Domain classes: rappresentano gli oggetti di interesse all'applicazione.

Controller classes: Si occupano della business logic dell'applicazione(cioè la logica applicativa che rende operativa l'applicazione, l'elaborazione).

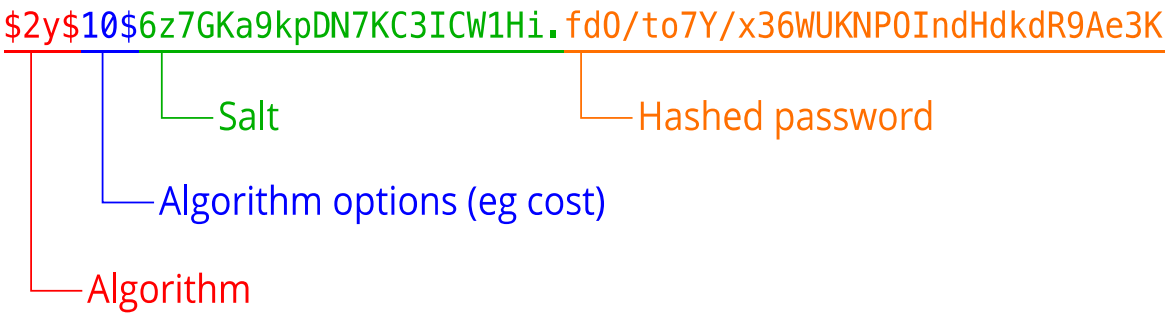
User Interface: Presenta i dati e interagisce con l'utente.

POLITICHE DI SICUREZZA

La password fornita dagli utenti viene salvata nel database previa crittografia, tramite la funzione

string **crypt** (string $str [, string $salt])

Il parametro "salt" verrà applicato durante il processo di hashing della password. Questo garantirà che la password criptata, memorizzata nel database, risulti diversa dalla versione crittografata della medesima password in un altro sito.
La funzione crypt( ) restituità una stringa criptata come illustrato nella figura.



La funzione sfrutta l'algoritmo standard di crittografia di UNIX basato su DES.

Si sono scartete le funzioni md5( ) e sha1( ), deprecate in quanto rapide ed efficienti, ma pertanto deboli di fronte ad attacchi di forza bruta.

Per prevenire attacchi di tipo SQL injection sono stati implementati una serie di metodi atti a verificare e pulire gli input degli utenti. Si è fatto uso dei principali metodi messi a disposizione da PHP per controllare le variabili $_POST e $_GET, quali:

- htmlspecialchars( ) : per formattare correttamente i dati prima di salvarli nel database.
- filter_var( ): per filtrare e ripulire gli input degli utenti.

Inoltre si è fatto uso di espressioni regolari per validare l'input.
Codice del file commons.php

```php
<?php


//pulisce i dati inseriti da utente
function puliziaDati ($stringa) {
     $stringa=addslashes (strip_tags(trim($stringa)));

     return $stringa;
}

function presentaDati ($stringa) {
     $stringa=htmlspecialchars(stripslashes ($stringa));

     return $stringa;
}



//controlla che un username sia valido
function usernameValido($username) {

     return preg_match('/^[a-zA-Z0-9]*_?[a-zA-Z0-0]*$/', $username);

}

//controlla che email sia valida
function emailValida ($email) {
     return (filter_var($email, FILTER_VALIDATE_EMAIL));
}
```

Per impedire l'iscrizione di bot al sito web ed evitare così attacchi di spam, si è ricorsi all'uso di una libreria **CAPTCHA** (Completely Automated Public Turing test to tell Computers and Humans Apart) in fase di registrazione utente. Si tratta di un test che richiede all'utente di scrivere lettere e numeri presenti in una sequenza distorta o offuscata, per poter completare la procedura di registrazione.

Tale libreria permette di generare codici CAPTCHA settando parametri relativi a fonts, numero di caratteri, numero di linee di disturbo, colori, ombreggiature, dimensioni dei font, set di caratteri.

Il cuore della libreria è nel metodo DrawCharacters( ), di cui riportiamo il codice

```php
 function DrawCharacters() {
         // loop through and write out selected number of characters
         for ($i = 0; $i < strlen($this->sCode); $i++) {
            // select random font
            $sCurrentFont = $this->aFonts[array_rand($this->aFonts)];

            // select random colour
```

```
            if ($this->bUseColour) {
               $iTextColour = imagecolorallocate($this->oImage, rand(0,
100), rand(0, 100), rand(0, 100));

               if ($this->bCharShadow) {
                  // shadow colour
                  $iShadowColour = imagecolorallocate($this->oImage, rand(0,
100), rand(0, 100), rand(0, 100));
               }
            } else {
               $iRandColour = rand(0, 100);
               $iTextColour = imagecolorallocate($this->oImage,
$iRandColour, $iRandColour, $iRandColour);

               if ($this->bCharShadow) {
                  // shadow colour
                  $iRandColour = rand(0, 100);
                  $iShadowColour = imagecolorallocate($this->oImage,
$iRandColour, $iRandColour, $iRandColour);
               }
            }

            // select random font size
            $iFontSize = rand($this->iMinFontSize, $this->iMaxFontSize);

            // select random angle
            $iAngle = rand(-30, 30);

            // get dimensions of character in selected font and text size
            $aCharDetails = imageftbbox($iFontSize, $iAngle, $sCurrentFont,
$this->sCode[$i], array());

            // calculate character starting coordinates
            $iX = $this->iSpacing / 4 + $i * $this->iSpacing;
            $iCharHeight = $aCharDetails[2] - $aCharDetails[5];
            $iY = $this->iHeight / 2 + $iCharHeight / 4;

            // write text to image
            imagefttext($this->oImage, $iFontSize, $iAngle, $iX, $iY,
$iTextColour, $sCurrentFont, $this->sCode[$i], array());

            if ($this->bCharShadow) {
               $iOffsetAngle = rand(-30, 30);

               $iRandOffsetX = rand(-5, 5);
               $iRandOffsetY = rand(-5, 5);

               imagefttext($this->oImage, $iFontSize, $iOffsetAngle, $iX +
$iRandOffsetX, $iY + $iRandOffsetY, $iShadowColour, $sCurrentFont, $this-
>sCode[$i], array());
            }
         }
      }
```

Il metodo disegna il codice basandosi su funzioni randomiche che alterano colori, font, dimensioni, angolazioni delle linee e così via.

COMUNICAZIONE CLIENT SERVER

Per la comunicazione tra il server e il client mobile dell'applicazione si è scelto di utilizzare JSON.
La struttura del messaggio sarà strutturata nel seguente modo in caso di successo:

```
{
        “tag” : “tag_operazione”,
        “success” : 1,
        “error”: 0,
        “dato1” : “valore1”,
        “dato2” : “valore2”
        …..
}
```

Dove il tag indica al server che tipo di operazione è stata richiesta (login, registrazione di un nuovo utente, registrazione di un nuovo insegnante, inserimento di un nuovo annuncio e così via),
il campo success indicherà con 1 l'effettivo successo e con 0 il fallimento dell'operazione.
Il campo error segnalerà l'eventuale presenza di errori. Infine i campi successivi saranno destinati ai dati veri e propri.
In caso di errore il messaggio avrà la seguente struttura:

```
{
        “tag” : “tag_operazione”,
        “success” : 0,
        “error”: codice_errore,
        “error_msg”: “Si è verificato il seguente errore”
}
```

In cui il campo “success” avrà valore 0 e il campo “error” avrà un codice numerico corrispondente al tipo di errore. Nel campo “error_msg” verrà inserito il messaggio d'errore.

Per il parsing della risposta JSON implementiamo la classe JSONParser.java, contenente il metodo getJSONFromUrl (String url, List<NameValuePair> params) che si occuperà di effettuare la connessione al server e di ricevere il messaggio dallo stream di input.

```
public class JSONParser {

	static InputStream is = null;
	static JSONObject jObj = null;
	static String json = "";

	//constructor
	public JSONParser() {

	}

	public JSONObject getJSONFromUrl(String url, List<NameValuePair> params) {

		// Making HTTP request
		try {
			// defaultHttpClient
			DefaultHttpClient httpClient = new DefaultHttpClient();
			HttpPost httpPost = new HttpPost(url);
			httpPost.setEntity(new UrlEncodedFormEntity(params,HTTP.UTF_8));

			HttpResponse httpResponse = httpClient.execute(httpPost);
			HttpEntity httpEntity = httpResponse.getEntity();
			is = httpEntity.getContent();
		}
		catch (UnsupportedEncodingException e) {
			e.printStackTrace();
		}
		catch (ClientProtocolException e){
			e.printStackTrace();
		}
		catch (IOException e) {
			e.printStackTrace();
		}

		try {
			BufferedReader reader = new BufferedReader(new InputStreamReader(is,
"iso-8859-1"), 8);
			StringBuilder sb = new StringBuilder();
			String line = null;
			while ((line = reader.readLine()) != null) {
				sb.append(line + "\n");
			}
			is.close();
			json = sb.toString();
			Log.e("JSON", json);

		}
		catch (Exception e) {
			Log.e("Buffer Error", "Error converting result " + e.toString());
		}

		//try parse  the string to a JSON object
		try {
			jObj = new JSONObject(json);
		}
		catch (JSONException e) {
			Log.e("JSON Parser", "Error parsing data " + e.toString());
```

```
            }

            //return JSON string
            return jObj;
        }

}
```

Il JSONParser verrà chiamato poi dai singoli metodi che implementano le varie funzioni di interazione col server, incapsulando nella lista di NameValuePair le richieste del client e restituendo il JSONObject di risposta del server.

Tutte le attività di interazione col server sono incapsulate in un AsynTask, poiché le operazioni di connessione sono lente e devono svolgersi in un thread separato da quello principale, per garantire una buona esperienza utente.

Poichè il server e il database usano una codifica di tipo UTF-8, si è reso necessario specificare tale codifica prima dell'invio delle richieste al server:

```
httpPost.setEntity(new UrlEncodedFormEntity(params,HTTP.UTF_8));
Così facendo si preservano invariate le parole accentate e i caratteri speciali.
```

APPLICAZIONE CLIENT ANDROID

La nostra applicazione sarà compatibile a partire dalla versione
Android 3.0, poiché faremo uso di alcuni costrutti peculiari di questa
versione, quali fragment, cursorLoader, ecc.

SERVICE

L'avvio di comunicazione col server viene delegato al RefreshService, che viene invocato durante l'uso dell'applicazione oppure può essere attivato manualmente dall'utente, premendo l'opzione di refresh dal menù action bar. Le attività su rete rendono il service il candidato ideale per implementarle. A differenza di una activity, un service non ha una interfaccia utente, ma è semplicemente una porzione di codice che viene eseguita in background. Per i nostri scopi, ovvero l'aggiornamento del database, in base alle informazioni reperite dal server, basterà un service di tipo unbound.

```
public class RefreshService extends IntentService {

	private static final String TAG = RefreshService.class.getSimpleName();

	public RefreshService() {
		super(TAG);
	}

	@Override
	public void onCreate() {
		super.onCreate();
		Log.d(TAG, "onCreated");
	}

	@Override
	protected void onHandleIntent(Intent intent){


		Log.d(TAG, "OnStarted");

ContentValues values = new ContentValues(); //contentValues è una semplice struttura di
dati che consiste in una
                                                                        //coppi
a nome-valore che mappano i nomi delle tabelle del db ai rispettivi valori

		AnnuncioFunctions af = new AnnuncioFunctions(); //
		try {
			int count = 0;
			int conto = 50;
			List<Annuncio> lista = af.readAll(conto);
```

```
                    if (!lista.isEmpty()) {
                          for (Annuncio ann : lista){
                                values.clear();
                                values.put(StatusContract.Column.ID, ann.getId());
                                values.put(StatusContract.Column.TITOLO, ann.getTitolo());
                                values.put(StatusContract.Column.CITTA, ann.getCitta());
                                values.put(StatusContract.Column.INDIRIZZO,
ann.getIndirizzo());
                                values.put(StatusContract.Column.INSEGNANTE,
ann.getInsegnante());
                                values.put(StatusContract.Column.MATERIA,
ann.getMateria());
                                values.put(StatusContract.Column.PREZZO, ann.getPrezzo());
                                values.put(StatusContract.Column.TESTO, ann.getTesto());
                                values.put(StatusContract.Column.NOMEINSEGNANTE,
ann.getNomeInsegnante());
                                values.put(StatusContract.Column.ELEMENTARE,
ann.getElementare());
                                values.put(StatusContract.Column.SUPERIORE,
ann.getSuperiore());
                                values.put(StatusContract.Column.UNIVERSITARIO,
ann.getUniversitario());



                                //db.insertWithOnConflict(StatusContract.TABLE, null,
values, SQLiteDatabase.CONFLICT_IGNORE);
                                Uri uri
=getContentResolver().insert(StatusContract.CONTENT_URI, values);
                                if (uri != null) {
                                      count++;
                                      //Log.d (....)
                                }
                          }
                    }
                    else{
                          //la lista è vuota, cioè non ci sono annunci
                    }

                    Log.d(TAG, "Il caricamento degli annunci è andato a buon fine");
              }
              catch (Exception e) {
                    Log.e(TAG, "Caricamento annunci dalla rete fallito", e);
                    e.printStackTrace();
              }

              try {
                    // ===============  carichiamo anche gli insegnanti ===============

                    ContentValues valuesDocenti = new ContentValues();

                    List<Docente> listaDoc = af.readDocenti();
                    if (!listaDoc.isEmpty()) {
                          for (Docente ann : listaDoc){
                                valuesDocenti.clear();
                                valuesDocenti.put(StatusContract.Column.ID, ann.getId());
                                valuesDocenti.put(StatusContract.Column.TITOLO,
ann.getTitolo());
                                valuesDocenti.put(StatusContract.Column.CITTA,
ann.getCitta());

                                valuesDocenti.put(StatusContract.Column.NOMEINSEGNANTE,
ann.getNome());
```

```
                                valuesDocenti.put(StatusContract.Column.TELEFONO,
ann.getTelefono());



                                //db.insertWithOnConflict(StatusContract.TABLE, null,
values, SQLiteDatabase.CONFLICT_IGNORE);
                                Uri uri =
getContentResolver().insert(StatusContract.URI_DOCENTI, valuesDocenti);
                                if (uri != null) {

                                      //Log.d (....)
                                }
                          }
                    }
                    else{
                          //la lista è vuota, cioè non ci sono docenti
                          Log.d(TAG, "La lista dei docenti risulato vuota");
                    }

              }
              catch (Exception e) {
                    Log.e(TAG, "Caricamento insegnanti dalla rete fallito", e);
                    e.printStackTrace();
              }

              return;
        }

        @Override
        public void onDestroy() {
              super.onDestroy();Log.d(TAG, "onDestroyd");
        }



}
```

Poiché il nostro sistema dispone di vari tipi di utente, si rende necessario realizzare un meccanismo che tenga traccia di una sorta di **sessione utente**. Pertanto si realizza un database, tramite SQLite, contenente la tabella TABLE_LOGIN, contenente l'id, lo username, la password, un campo IS_INSEGNANTE e un campo AUTHENTICATED. Questa tabella viene riempita nel momento in cui si effettua il login con il server e si riceve esito positivo, con i dati ottenuti dal parsing del JSONObject di risposta del server.
In base ai dati contenuti nel database, l'activity DashboardActivity mosterà il layout appropriato per l'utente.

```
public class DashboardActivity extends Activity {

        AnnuncioFunctions function;
        Button btnLogout;
        Button btnNuovo;
        Button btnRegProf;

        @Override
```

```
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//Check login status in database
				function = new AnnuncioFunctions();

				//prima facciamo il check che sia un insegnante
				if (function.isInsegnante(getApplicationContext())) {
					//user already logged in and he's a teacher

					setContentView(R.layout.activity_dashboard_prof);

					// button di nuovo annuncio
					btnNuovo = (Button) findViewById(R.id.btn_nuovo_annuncio);
					btnNuovo.setOnClickListener(new View.OnClickListener() {

						@Override
						public void onClick(View v) {
							Intent nuovoAnnuncio = new
Intent(getApplicationContext(), StatusActivity.class);
							startActivity(nuovoAnnuncio);
						}
					});

					//button logout
					btnLogout = (Button) findViewById(R.id.btnLogout_prof);

					btnLogout.setOnClickListener(new View.OnClickListener() {

						public void onClick(View arg0) {

							function.logoutUser(getApplicationContext());
							Intent home = new
Intent(getApplicationContext(),HomeActivity.class);
							home.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
							startActivity(home);
							//Closing dashboard screen
							finish();

						}
					});



				}
				//l'utente è loggato ma è un utente semplice
				else if (function.isUserLoggedIn(getApplicationContext())) {
					//user already logged in show databoard
					setContentView(R.layout.activity_dashboard);

					//butto registrati come insegnante
					btnRegProf = (Button) findViewById(R.id.buttonInsegnante);
					btnRegProf.setOnClickListener(new View.OnClickListener() {

						@Override
						public void onClick(View v) {

							Intent reg = new
Intent(getApplicationContext(), RegisterProfActivity.class);
							startActivity(reg);
							finish();

						}
					});
```

```
                                    //butto Logout

                                    btnLogout = (Button) findViewById(R.id.btnLogout);

                                    btnLogout.setOnClickListener(new View.OnClickListener() {

                                          public void onClick(View arg0) {

                                                function.logoutUser(getApplicationContext());
                                                Intent home = new
Intent(getApplicationContext(),HomeActivity.class);
                                                home.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                                startActivity(home);
                                                //Closing dashboard screen
                                                finish();

                                          }
                                    });

                              }

                              else {
                                    //user is not logged in, show login screen
                                    Intent home = new
Intent(getApplicationContext(),HomeActivity.class);
                                    home.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                    startActivity(home);
                                    //Closing dashboard screen
                                    finish();
                              }



      }



}
```

Per utilità creiamo una classe di **metadati**, in cui definiremo il nome del DB, delle diverse tabelle e delle colonne.

File StatusContract.java

```
public class StatusContract {

      //DB specific constants
      public static final String DB_NAME = "timeline.db"; //il file SQLite che conterrà il
db
      public static final int DB_VERSION = 1;
      public static final String TABLE = "annuncio";
      //
      public static final String TABLE_DOCENTI = "docente";

      public static final String TABLE_LOGIN = "login";

      public static final String IS_INSEGNANTE_TRUE = "1"; //l'utente è un insegnante?
      public static final String IS_AUTH_TRUE = "1"; //l'utente è autenticato?

      public static final String AUTHORITY = "com.alice.alearning.AnnuncioProvider";
```

```
	public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" +
TABLE);
	//
	public static final Uri URI_DOCENTI = Uri.parse("content://" + AUTHORITY + "/" +
TABLE_DOCENTI);
	public static final int STATUS_ITEM = 1;
	public static final int STATUS_DIR = 2;
	public static final int DOC_ITEM = 3;
	public static final int DOC_DIR = 4;
	public static final String STATUS_TYPE_ITEM =
"vnd.android.cursor.item/vnd.com.alice.alearning.provider.annuncio";
	public static final String STATUS_TYPE_DIR =
"vnd.android.cursor.dir/vnd.com.alice.alearning.provider.annuncio";

	public static final String DOC_TYPE_ITEM =
"vnd.android.cursor.item/vnd.com.alice.alearning.provider.docente";
	public static final String DOC_TYPE_DIR =
"vnd.android.cursor.dir/vnd.com.alice.alearning.provider.docente";

	public class Column {
		public static final String ID = BaseColumns._ID;
		public static final String INSEGNANTE = "insegnante";
		public static final String PREZZO = "prezzo";
		public static final String TESTO = "testo";
		public static final String TITOLO = "titolo";
		public static final String MATERIA = "materia";
		public static final String CITTA = "citta";
		public static final String INDIRIZZO = "indirizzo";
		public static final String NOMEINSEGNANTE = "nomeInsegnante";
		public static final String ELEMENTARE = "elementare";
		public static final String SUPERIORE = "superiore";
		public static final String UNIVERSITARIO = "universitario";
		public static final String TELEFONO = "telefono";

		//column for login table
		public static final String USERNAME = "username";
		//id lo riciclo
		public static final String PASSWORD = "password";
		public static final String ISINSEGNANTE = "isinsegnante";
		public static final String AUTHENTICATED = "authenticated";



	}

}
```

CURSORLOADER

Per comodità di accesso ai dati, la struttura del database client sarà leggermente diversa rispetto a quella del server. Si è operato un accorpamento dei dati per limitare l'interazione con il server e avere comodamente a disposizioni quanti più dati possibili nella medesima tabella. Nello specifico si sono accorpate le tabelle annuncio e livelloAnnunci.
Per i dati realitivi agli annunci si è deciso di fare uso di un Content Provider, poiché esso è necessario per invocare il costruttore di **CursorLoader**.

I Loader, introdotti dalla versione Android 3.0 (API level 11), servono principalmente a caricare dati in modo asincrono in un thread separato da quello della UI, grazie all'uso di AsyncTask.
Il CursorLoader interroga il Content Resolver in background (in modo da non bloccare la User Interface e garantire una buona esperienza utente) e impone allo sviluppatore di non chiudere il cursore.
Il loader assicura la persistenza del fetch dei dati, in modo tale da evitare operazioni ripetitive di fetch (ad esempio in caso di rotazione dello schermo).
La caratteristica più rilevante del CursorLoader, che lo rende migliore di un Loader, è la capacità di aggiornarsi in caso di aggiornamento dei dati, riconnettendosi automaticamente e auto-aggiornandosi, senza la necessità di fare una nuova query al cursore.

File TimelineFragment.java

```
public class TimelineFragment extends ListFragment implements LoaderCallbacks<Cursor> {
	private static final String TAG = TimelineFragment.class.getSimpleName();

	private static final String [] FROM = {StatusContract.Column.MATERIA,
StatusContract.Column.TESTO, StatusContract.Column.TITOLO};

	private static final int [] TO = { R.id.list_item_text_materia,
R.id.list_item_text_testo, R.id.lista_item_text_titolo };

	private static final int LOADER_ID = 42;
	private SimpleCursorAdapter mAdapter;



	@Override
	public void onActivityCreated (Bundle savedInstanceState) {
		super.onActivityCreated(savedInstanceState);

		mAdapter = new SimpleCursorAdapter(getActivity(), R.layout.list_item, null,
FROM, TO, 0);


		setListAdapter(mAdapter);
		getLoaderManager().initLoader(LOADER_ID, null, this);

	}

	@Override
	public void onListItemClick(ListView l, View v, int position, long id) {
		/*
		//Get the details fragment
		DetailsFragment fragment = (DetailsFragment) getFragmentManager()
				.findFragmentById(R.id.fragment_details);

		//Is details fragment visible?
```

```
            if (fragment != null && fragment.isVisible()) {
                  fragment.updateView(id);
            }
            else {
                  startActivity(new Intent(getActivity(), DetailsActivity.class)
                        .putExtra(StatusContract.Column.ID, id));
            }
            */


            startActivity(new Intent(getActivity(), DetailsActivity.class)
            .putExtra(StatusContract.Column.ID, id));


      }


      // ----- Loader Callback ------
      @Override
      public Loader<Cursor> onCreateLoader(int id, Bundle args) {
            if (id != LOADER_ID)
                  return null;
            Log.d(TAG, "onCreateLoader");

            return new CursorLoader(getActivity(), StatusContract.CONTENT_URI, null, null,
null,null);

      }

      @Override
      public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
            //Get the details fragment
            DetailsFragment fragment = (DetailsFragment) getFragmentManager()
                        .findFragmentById(R.id.fragment_details);

            //Is details fragment visible?
            if (fragment != null && fragment.isVisible() && cursor.getCount() == 0) {
                  fragment.updateView(-1);
                  Toast.makeText(getActivity(), "no data", Toast.LENGTH_LONG).show();
            }

            Log.d(TAG, "onLoadFinished with cursor: "+ cursor.getCount());
            mAdapter.swapCursor(cursor);
      }


      @Override
      public void onLoaderReset(Loader<Cursor> loader) {
            mAdapter.swapCursor(null);
      }

}
```

CONTENT PROVIDER

Il content provider è una delle fondamenta di android. Esso permette di centralizzare il contenuto di una posizione e far sì che diverse applicazioni vi accedano. Il content provider è semplicemente una interfaccia per i dati, pertanto

non contempla l'effettiva memorizzazione. Nel nostro caso il content provider è realizzato nella classe AnnuncioProvider, che estende la classe di sistema ContentProvider e ne implementa i metodi non usati, come insert(), update(), delete(), query(), getType() e va dichiarato nel manifest della applicazione.
Gli oggetti all'interno di una singola app condividono un address space , che non è riconosciuto però dagli oggetti presenti in app diverse. Si usa pertanto un Uniform Resource Identifier (URI) una stringa che identifica una specifica risorsa, per localizzare un content provider, così composta:

content://com.alice.alearning.AnnuncioProvider/annuncio/21

dove:
“content://” assume sempre questo valore e sta ad indicare un provider.
“com.alice.alearning.AnnuncioProvider” è l'authority, che caratterizza in modo univoco il particolare content provider e comparirà anche nel manifest
“annuncio” indica il tipo di dati che il provider è in grado di fornire
“21” è l'ID dell'elemento specifico che stiamo richiamando. Se non specificato indicheremo tutto il set di dati.

Un content provider deve restituire il MIME type dei dati che ritorna. La prima parte del MIME è definita come:

“vnd.android.cursor.item/vnd.com.alice.alearning.provider.annuncio” nel caso di un elemento specifico. Oppure:
“vnd.android.cursor.dir/vnd.com.alice.alearning.provider.annuncio” nel caso il type rappresenti tutti gli elementi.

Per definire il tipo implementiamo un metodo getType(Uri uri) , che usa un'istanza della classe UriMatcher per restituire il MIME type appropriato.

Nell'uso di content provider si è deciso di non effettuare il close( ) sul cursore, come consigliato da Dianne Hackborn (Android framework engineer), poiché esso verrà chiuso in ogni caso alla chiusura del processo, qualora il kernel dovesse fare pulizia.

DBHELPER

La classe DbHelper sarà una estensione di SQLiteOpenHelper e conterrà tutti i metodi necessari alla gestione del database

```
public class DbHelper extends SQLiteOpenHelper {
	private static final String TAG = DbHelper.class.getSimpleName();
```

```
	//passiamo al costruttore il nome del database e il numero di versione
	public DbHelper(Context context) {
		super(context, StatusContract.DB_NAME, null, StatusContract.DB_VERSION);
	}

	//called only once first time we create the database
	@Override
	public void onCreate(SQLiteDatabase db) {
		//creiamo la tabella con una stringa di sql formata dalle costanti definite
prima
		//le %s sono le diciture di unix
		String sql = String.format("create table %s (%s int primary key, %s text, %s
text, %s text, %s text, %s text, %s float, %s text, %s text, %s text, %s text )",
				StatusContract.TABLE,
				StatusContract.Column.ID,
				StatusContract.Column.INSEGNANTE,
				StatusContract.Column.TESTO,
				StatusContract.Column.TITOLO,
				StatusContract.Column.MATERIA,
				StatusContract.Column.CITTA,
				StatusContract.Column.INDIRIZZO,
				StatusContract.Column.PREZZO,
				StatusContract.Column.NOMEINSEGNANTE,
				StatusContract.Column.ELEMENTARE,
				StatusContract.Column.SUPERIORE,
				StatusContract.Column.UNIVERSITARIO);

		//creiamo la tabella per docente
		String sql_docente = String.format("create table %s (%s int primary key, %s
text, %s text, %s text, %s text )",
				StatusContract.TABLE_DOCENTI,
				StatusContract.Column.ID,
				StatusContract.Column.NOMEINSEGNANTE,
				StatusContract.Column.TITOLO,
				StatusContract.Column.CITTA,
				StatusContract.Column.TELEFONO);

		//creiamo la tabella per login
		String sql_login = String.format("create table %s (%s int primary key, %s
text, %s text, %s text, %s text)",
				StatusContract.TABLE_LOGIN,
				StatusContract.Column.ID,
				StatusContract.Column.USERNAME,
				StatusContract.Column.PASSWORD,
				StatusContract.Column.ISINSEGNANTE,
				StatusContract.Column.AUTHENTICATED);


		Log.d(TAG, "onCreate with SQL: "+sql);
		db.execSQL(sql);
		Log.d(TAG, "onCreat with SQL: " +sql_docente);
		db.execSQL(sql_docente);
		Log.d(TAG, "onCreat with SQL: " +sql_login);
		db.execSQL(sql_login);
	}

	//Gets called whenever existing version != new version, i.e. schema changed

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

		//Typically you do ALTER TABLE...
		db.execSQL("drop table if exists "+ StatusContract.TABLE);
		db.execSQL("drop table if exists "+ StatusContract.TABLE_DOCENTI);
```

```
		db.execSQL("drop table if exists "+ StatusContract.TABLE_LOGIN);
		onCreate(db);
	}

	public Cursor user() {
		String query = "SELECT * FROM " + StatusContract.TABLE_LOGIN;
		SQLiteDatabase db = this.getReadableDatabase();
		Cursor cursor = db.rawQuery (query, null);
		return cursor;
	}

	public boolean isLogged() {
		String query = "SELECT "+ StatusContract.Column.AUTHENTICATED + " FROM " +
StatusContract.TABLE_LOGIN;
		SQLiteDatabase db = this.getReadableDatabase();
		Cursor cursor = db.rawQuery (query, null);
		if (!cursor.moveToFirst())
			return false;
		else {
			String authenticated =
cursor.getString(cursor.getColumnIndex(StatusContract.Column.AUTHENTICATED));
			if (authenticated == null) return false;
			else {
				if (authenticated.equals(StatusContract.IS_AUTH_TRUE)) return
true;
				else return false;
			}
		}
	}

	public boolean isInsegnante() {
		String query = "SELECT "+ StatusContract.Column.ISINSEGNANTE + " FROM " +
StatusContract.TABLE_LOGIN;
		SQLiteDatabase db = this.getReadableDatabase();
		Cursor cursor = db.rawQuery (query, null);
		if (!cursor.moveToFirst())
			return false;
		else {
			String isInsegnante =
cursor.getString(cursor.getColumnIndex(StatusContract.Column.ISINSEGNANTE));
			if (isInsegnante == null) return false;
			else {
				if (isInsegnante.equals(StatusContract.IS_INSEGNANTE_TRUE))
return true;
				else return false;
			}
		}
	}

	public void resetLoginTable() {
		SQLiteDatabase db = this.getWritableDatabase();
		//Delete All rows
		db.delete(StatusContract.TABLE_LOGIN,  null, null);
		db.close();
	}

	public void addUser(String username, String password, int id, String authenticated,
String isInsegnante)  {
		SQLiteDatabase db = this.getWritableDatabase();

		ContentValues values = new ContentValues();
		values.put(StatusContract.Column.USERNAME, username); //nome
		values.put(StatusContract.Column.PASSWORD, password); //password
		values.put(StatusContract.Column.ID, id);
```

```
            values.put(StatusContract.Column.AUTHENTICATED, authenticated);
            values.put(StatusContract.Column.ISINSEGNANTE, isInsegnante);

            //inserting row
            db.insert(StatusContract.TABLE_LOGIN,  null, values);
            db.close(); //Closing database connection

      }

      public void diventaInsegnante(String id)  {
            SQLiteDatabase db = this.getWritableDatabase();

            ContentValues values = new ContentValues();

            values.put(StatusContract.Column.ISINSEGNANTE,
StatusContract.IS_INSEGNANTE_TRUE);
            String where = StatusContract.Column.ID + " = ?";
            String whereArgs [] = new String [] {id};

            //inserting row
            db.update(StatusContract.TABLE_LOGIN,  values,  where , whereArgs);
            db.close(); //Closing database connection

      }

      public Cursor queryAnnuncio(String query) {

            SQLiteDatabase db = this.getReadableDatabase();
            Cursor cursor = db.rawQuery (query, null);
            return cursor;
      }

      public long getId() {
            String query = "SELECT "+ StatusContract.Column.ID + " FROM " +
StatusContract.TABLE_LOGIN;
            SQLiteDatabase db = this.getReadableDatabase();
            Cursor cursor = db.rawQuery (query, null);
            if (!cursor.moveToFirst())
                  return -1;
            else {
                  long id =
cursor.getLong(cursor.getColumnIndex(StatusContract.Column.ID));
                  return id;

            }

      }


      public int delete(String id) {
            SQLiteDatabase db = this.getWritableDatabase();

            String where = StatusContract.Column.ID + " = ?";
            String whereArgs [] = new String [] {id};

            //deleting row
            int r = db.delete(StatusContract.TABLE, where , whereArgs);
            db.close(); //Closing database connection
            return r;
      }



}
```

MENU'

Per garantire all'utente usabilità e rapidità di azione, si è provveduto ad implementare una **action bar** contenente un menù per l'accesso rapido alla funzionalità di ricerca e la funzionalità di refresh dei dati dal server.

```
	//menu

	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) { //gestiamo l'iterazione utente con il menu

		switch (item.getItemId()) { //Che elemento ha cliccato l'utente? Per scoprirlo recuperiamo l'id e
									//facciamo un controllo switch case per sapere quale opzione del menu è stata scelta
		case R.id.action_refresh:
			startService(new Intent(this, RefreshService.class));
			return true;
		case R.id.action_search:
			Intent ricerca = new Intent(getApplicationContext(), RicercaActivity.class);
			startActivity(ricerca);
			return true;

		default:
			return false;
		}
	}
```

GOOGLE MAPS

Si è dotata la applicazione delle Google Maps API v2, attraverso l'aggiunta della libreria specifica dei servizi Google play e l'inserimento nel file manifest degli opportuni permessi:

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

```
    <!-- add for map2 -->
    <permission
        android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />

    <uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />

     <!-- Maps API needs OpenGL ES 2.0. -->
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />
```

Per chiamare la mappa vera e propria si è fatto uso dei Google Maps Intent, per visualizzare la locazione esatta dell'annuncio:

```
      textIndirizzo.setOnClickListener(new View.OnClickListener() {

                  public void onClick(View arg0){
                        Uri gmmIntentUri = Uri.parse("geo:0,0?q="+indirizzo+", "+citta+",
Italia");
                        Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
                        mapIntent.setPackage("com.google.android.apps.maps");
                        startActivity(mapIntent);
                  }

            });
```

SUPPORTO PER SCHERMI DI VARIE DIMENSIONI

Per supportare schermi di varia dimensione si è fatto uso di Fragment, un nuovo approccio per l'interfaccia utente, introdotto a partire da Android 3.0, che separa il container dell'activity dalla UI. Questo rende l'applicazione più responsive. Riportiamo l'esempio del layout relativo al singolo docente. Visualizzerò un fragment destinato ai dati dell'insegnante e un secondo fragment che visualizzi l'elenco degli annunci relativi ad esso.

```
  <!-- Docenti Fragment -->
    <fragment
        android:id="@+id/fragment_docenti"
        android:name="com.alice.alearning.DocentiFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerHorizontal="true" />

    <!-- Details fragment -->
```

```
<fragment
    android:id="@+id/fragment_detailsdoc"
    android:name="com.alice.alearning.DetailsDocenteFragment"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    tools:layout="@layout/list_item" />
```

Per favorire una eventuale internalizzazione della nostra applicazione, le stringhe visualizzate dall'utente sono mantenute nel file values>strings che potrà essere specializzato in base alla nazionalità.

Manuale e il test su macchina virtuale

Qualora si volesse testare il server localhost su macchina virtuale, sarà necessario avviare il server apache e il database da riga di comando:

sudo /opt/lampp/lampp start

Per effettuare l'accesso da amministratore immettere le credenziali: holyna - faraone

BIBLIOGRAFIA

Sviluppare siti dinamici con PHP e MySQL, Andrea Tarr
Sviluppare applicazioni con PHP e MySQL, Kevin Yank
Android 4 guida per lo sviluppatore, Massimo Carli
Sviluppare con Android, Marko Gargenta e Masumi Nakamura
Materiale didattico del corso di Base di dati, Maurizio Lenzerini
Materiale didattico del corso di Progettazione software, Giuseppe De Giacomo
Materiale didattico del corso di Progetto di applicazione software, Massimo Mecella
Materiale didattico del corso di Progetto di applicazione software, Antonella Poggi
Manuale di PHP online, PHP.net
Android JSON parsing tutorial, androidhive.info
Android App Development with Java Essential Training, lynda.com