TESINA TECNICHE AUDIOVISIVE

PROGETTO: APP MOBILE ANDROID

PROF. ORLANDI

PROF. PROIETTI

REALIZZATA DA:

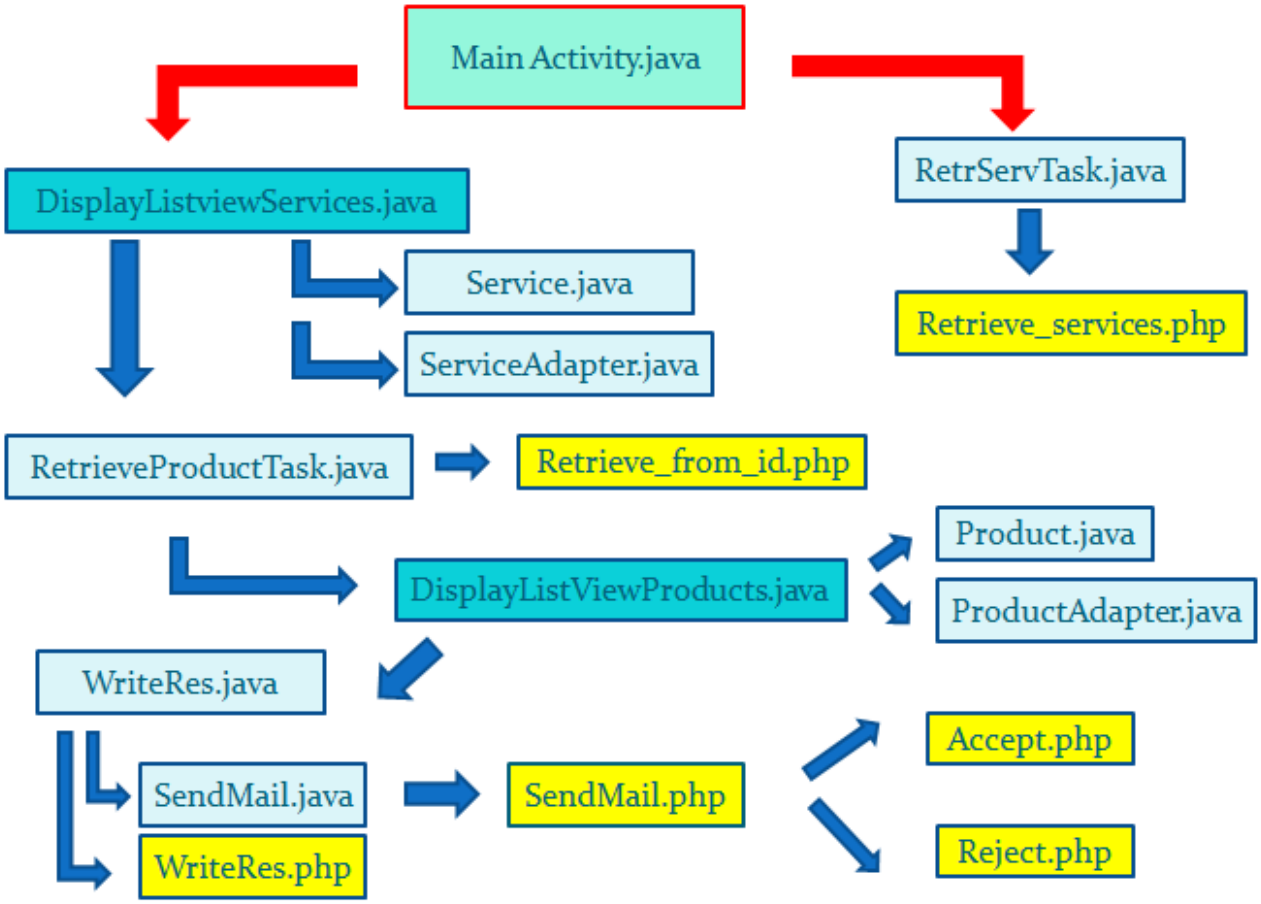
CAMPO MANUEL

GALDIERI ROBERTO

INDICE

INTRODUZIONE

Obiettivo di questo lavoro, è stato quello di creare un'applicazione Android con la quale un utente è in grado di prenotare dei prodotti, comunicando tramite un server con un destinatario, il quale risponde tramite mail e può decidere se accettare o rifiutare la richiesta di prenotazione.



La mappa concettuale mostra le relazioni esistenti tra tutti gli elementi che formano la struttura della nostra app.

All'interno di tale struttura, la coesistenza di file .java e .php ci consente di avere un'app di tipo 'dinamico', in grado dunque di potersi connettere ad un DataBase ed eseguire le proprie funzioni comunicando di volta in volta con esso.

PARTE I – ALTERVISTA

- CREAZIONE DATABASE
- DESCRIZIONE TABELLE

Come primo passo abbiamo creato il nostro database remoto su Altervista contenente tre tabelle:

1) Servizi
2) Prodotti
3) Prenotazioni

La prima tabella elenca i Servizi offerti da un gestore caratterizzata dai seguenti attributi: id, nome, descrizione, indirizzo e mail (con id chiave primaria).

	id	nome	descr	indirizzo	mail
Modifica Copia Elimina	1	Ristorazione	Menu	Caraibe	tizio
Modifica Copia Elimina	2	Cocktail	Alcolici	Bancone	Caio
Modifica Copia Elimina	3	Tavoli	Compleanni	Divani	Sempronio

La tabella Prodotti mostra gli Oggetti in relazione ai rispettivi servizi.

I campi che costituiscono questa tabella sono: id, idOgg, nome, descrizione, prezzo.

Abbiamo ottenuto la corrispondenza tra le tabelle Servizi e Prodotti usando una doppia chiave primaria: id e idOgg. La prima identifica il servizio selezionato, la seconda risulta essere la chiave univoca del Prodotto.

	id	idOg	nome	descriz	prezzo
Modifica Copia Elimina	1	1	Pizzero	pizza	10.00
Modifica Copia Elimina	1	2	Grillero	Carne	15.00
Modifica Copia Elimina	2	3	Cuba Libre	Dolce	8.50
Modifica Copia Elimina	2	4	Moijto	Secco	8.50
Modifica Copia Elimina	3	5	Divano	Largo	20.00
Modifica Copia Elimina	3	6	Prive	Stretto	20.00

Infine, la tabella ‘Prenotazioni’ è caratterizzata dai campi timestamp, id del servizio, lista dei prodotti prenotati, numero degli oggetti prenotati, stato prenotazione e mail dell’utente (con timestamp chiave primaria).

timestamp	id_serv	lista_oggetti_pr	numero_oggetti_pr	stato	email
1468486002107	1	[Pizzero;Grillero]	[4;7]	rifiutata	manuel.campo90@gmail.com
1468486176744	1	[Pizzero;Grillero]	[2;3]	rifiutata	manuel.campo90@gmail.com
1468488247954	1	[Pizzero;Grillero]	[1;1]	rifiutata	manuel.campo90@gmail.com
1468488362644	1	[Pizzero;Grillero]	[6;15]	accettata	manuel.campo90@gmail.com
1468488615278	1	[Pizzero;Grillero]	[3;4]	rifiutata	manuel.campo90@gmail.com
1468491013166	3	[Divano;Prive]	[8;3]	accettata	galdieri.roberto@gmail.com
1468491166577	2	[Cuba Libre;Moijto]	[9;6]	in attesa	galdieri.roberto@gmail.com
1468491917584	3	[Divano;Prive]	[5;4]	accettata	manuel.campo90@gmail.com

Quest’ultima tabella risulta essere in continuo aggiornamento: ogni volta che un utente prenota uno o più prodotti, viene aggiunto un record con stato impostato “IN ATTESA”. Non appena il gestore accetta o meno la prenotazione, lo stato viene settato rispettivamente in “ACCETTATA” o “RIFIUTATA”.

PARTE II – ANDROID E PHP

- RECUPERO DELLA LISTA DEI SERVIZI
- CREAZIONE LISTVIEW SERVIZI

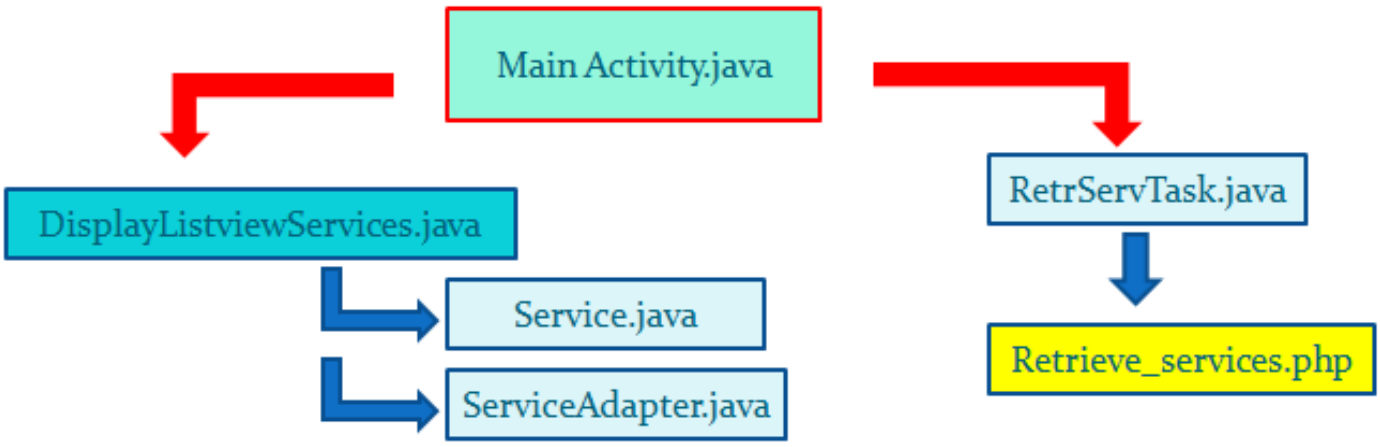
Per recuperare i Servizi e i rispettivi Prodotti dal DB e mostrarli a schermo sulla nostra App tramite delle ListView, occorrono dei file che:

1) Creino lo scheletro della lista
2) Richiedano al DB, attraverso un'interrogazione, gli elementi necessari a riempire la lista
3) Inseriscano questi all'interno di ogni riga

Il primo passo è dunque quello di richiedere al DB i Servizi, e mostrarli a schermo tramite ListView. Per fare questo, sono stati utilizzati i seguenti script spiegati in seguito nel dettaglio:

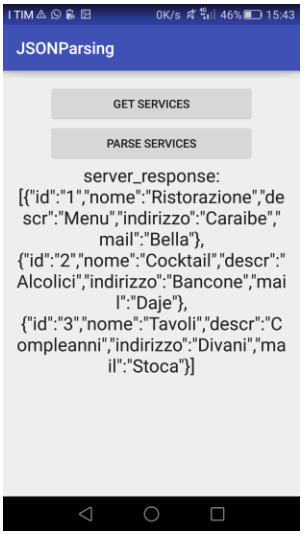
1) *MainActivity.java*
2) *RetrievingServicesTask.java*
3) *retrieve_services.php*
4) *DisplayListViewServices.java*
5) *Service.java*
6) *ServiceAdapter.java*

Essi sono legati tra loro come mostrato in figura:



MainActivity.java

E' lo script che gestisce l'activity iniziale della app ed è caratterizzata da due bottoni ('GET SERVICES' e 'PARSE SERVICES') che mi permettono di verificare la connessione al DB mediante il download di un file json.



Lo script contiene i due metodi che vengono richiamati al click dei rispettivi bottoni:

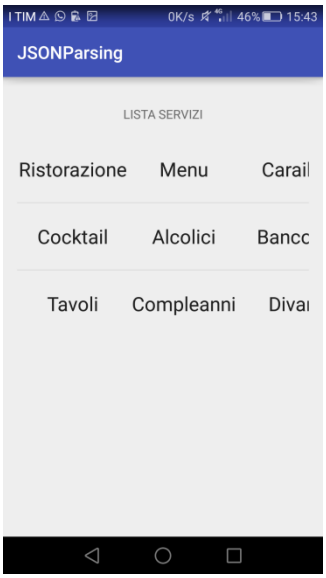
1) GetJSON: richiama al suo interno l'esecuzione del file RetrievingServicesTask.java

parseJSON: prende il JSON fornito dal metodo getJSON (contenente la risposta del database alla richiesta della lista dei servizi) e apre la nuova activity con la ListView di tutti i Servizi.

RetrievingServicesTask.java

Fa un interrogazione a database tramite lo script ***retrieve_services.php*** il quale accede al DB e richiede tutti i Servizi disponibili nella tabella 'Servizi', codificando poi il tutto in un file JSON; quest'ultimo viene salvato in una variabile di tipo stringa che viene passata al file MainActivity.java, quindi mandata allo script DisplayListViewServices.java.

DisplayListViewServices.java



È lo script che crea la ListView e ci mostra i Servizi con relative informazioni. Una ListView in Android ha bisogno di:

1) un layout per la singola riga (*services_row_layout*)
2) un Adapter che viene implementato all'interno di essa.

L'Adapter è la componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi per ogni elemento della ListView. L'Adapter è creato dal metodo setAdapter all'interno di questo script:

```
serviceAdapter = new ServiceAdapter( this, R.layout.product_row_layout );
listView.setAdapter( serviceAdapter );
json_string = getIntent().getExtras().getString( "json_data" );
```

A partire dalla stringa JSON viene creato un array contenente un elemento per ogni posizione, il quale viene poi scansionato tramite ciclo while: ad ogni passo del ciclo viene creata e riempita una riga della lista contenente: nome, descrizione, e indirizzo. Lo script fa uso di un costruttore (Service.java) al quale vengono passati i parametri salvati, in modo da ultimare il riempimento della singola riga. Il processo termina nel momento in cui sono stati scansionati tutti gli elementi dell'array.

Infine il metodo sendService ci permette, a seconda del servizio che selezioniamo, di avere la lista dei Prodotti di quel Servizio. La lista degli oggetti è creata dal file RetrievingProductsTask.java che interroga il DB tramite php.

Service.java

Costruttore che ci permette di descrivere un servizio tramite determinati parametri quali:

1) Nome
2) Descrizione
3) Indirizzo
4) Mail
5) id

All'interno di esso sono definiti i metodi get e set per ogni variabile.

ServiceAdapter.java

È caratterizzato da più metodi usati per il riempimento delle righe:

1) void add(Service object): dato un oggetto Servizio in input, lo aggiunge alla lista
2) int getCount(): restituisce la dimensione della lista
3) Object getItem(int position): dato un intero, ci restituisce l'oggetto in quella posizione

Creo la classe ServiceHolder che ci permette di salvare i parametri da usare nel TextView per ogni riga (nome, descrizione, indirizzo). Definisco la variabile serviceHolder per raccogliere i dati. Attraverso l'Inflater siamo in grado di convertire un layout di un file XML in una viewGroup corrispondente.

- ACQUISIZIONE PRODOTTI
- CREAZIONE LISTVIEW PRODOTTI

Una volta creata la ListView contenente i Servizi, dobbiamo essere in grado di ottenere i Prodotti del Servizio al quale siamo interessati.

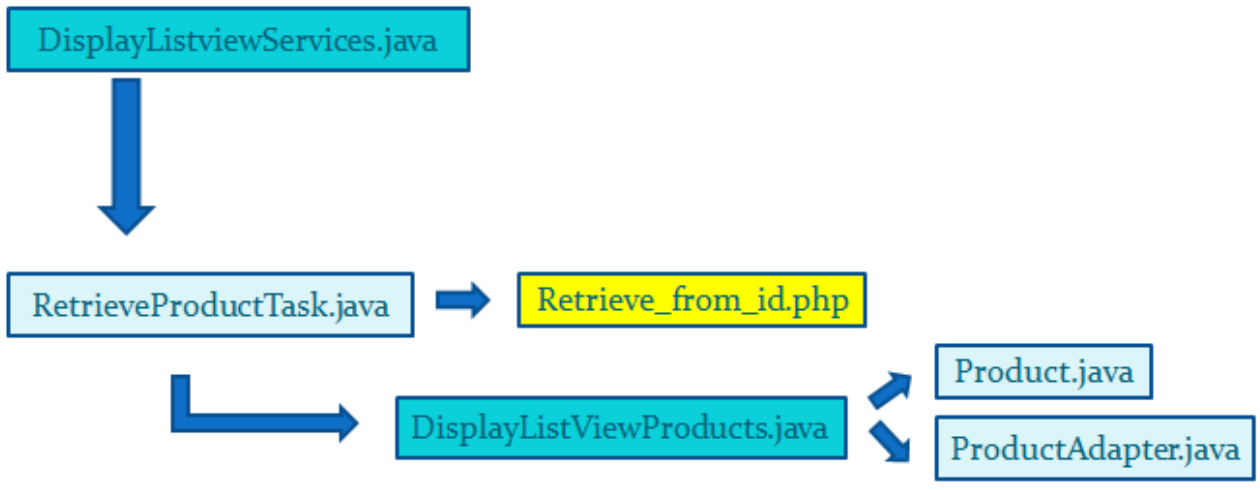
Occorre dunque:

1) Uno script .php che, a seconda del servizio che scegliamo, ricavi dal DB i Prodotti appartenenti a quel servizio
2) Una ListView che ci permetta di 'ospitare' i Prodotti trovati

Per questo secondo passo, sono stati realizzati gli script:

1) RetrievingProductsTask.java
2) Retrieve_from_id.php
3) DisplayListViewProduct.java
4) Product.java
5) ProductAdapter.java

Nella figura seguente possiamo osservare i legami di connessione esistenti tra le diverse componenti:



RetrievingProductsTask.java

Viene inizialmente definito il costruttore associato a questa classe con parametri in input id_serv, name, descr, indir e act. A seconda del parametro id_serv, vengono chiesti al DB i prodotti di quel determinato servizio.

L’interrogazione a DB è realizzata tramite il file php *retrieve_from_id.php*, seguito dal parametro “id_serv” passato in input al costruttore. In questo modo riusciamo in modo dinamico, attraverso un solo file php, a richiedere i prodotti dei vari servizi indipendentemente dal servizio scelto.

```
@Override
protected void onPreExecute() {
    json_url = "http://servizieee.altervista.org/php/retrieve_from_id.php?id=" + id_serv;
}
```

Nella parte finale, il metodo onPostExecute(String result) crea un Intent per aprire una nuova activity nella quale mostrare la lista dei prodotti e lo fa richiamando lo script DisplayListViewProducts.java

```
@Override
protected void onPostExecute( String result ) {
    Log.d( "DEBUG", result );
    Intent intent = new Intent ( act, DisplayListViewProducts.class );
    intent.putExtra( "json_data", result );
    intent.putExtra( "name", name );
    intent.putExtra( "descr", descr );
    intent.putExtra( "indir", indir );
    intent.putExtra( "id_serv", id_serv );
    act.startActivity( intent );
}
```

Retrieve_from_id.php

Ha il compito di richiedere al DB i prodotti di un determinato servizio, l’id del quale è inviato tramite URL dal file RetrievingProductsTask.java:

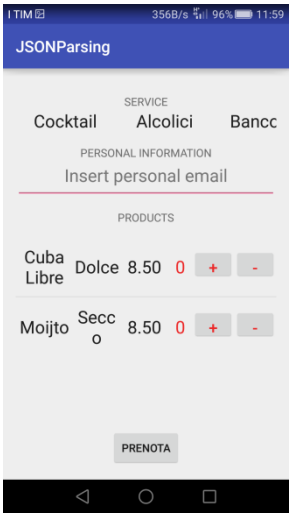
```
@Override
protected void onPreExecute() {
    json_url = "http://servizieee.altervista.org/php/retrieve_from_id.php?id=" + id_serv;
}
```

All’interno del file php, il parametro “id” è stato definito tramite metodo GET.

In tal modo, passando il parametro all’interno della query, viene creato un file JSON da inviare ad Android che in seguito creerà la ListView tramite i file DisplayListViewProduct.java e ProductAdapter.java.

```php
$input_from_android = $_GET["id"];
$sql = "SELECT nome, descr, indirizzo, mail FROM services WHERE id=$input_from_android";
$result = $conn->query($sql);
    while($row = $result->fetch_assoc()) {
        $output[]=$row;
    }
    echo "server_response:", json_encode($output);
```

DisplayListViewProduct.java



Viene mostrata a schermo la Lista dei Prodotti di un determinato servizio con relativi:

1) Nome
2) Descrizione
3) Prezzo
4) Quantità da Prenotare

È inoltre presente un tasto “PRENOTA” in fondo alla activity, il quale permette di effettuare l’ordine tramite una mail inviata al gestore che dovrà decidere se accettare o meno (spiegato meglio nel seguito).

Per comunicare l’esito della prenotazione, l’utente è tenuto ad inserire la propria mail sulla quale ricevere la risposta da parte del gestore.

Come nel caso della ListView dei Servizi, anche in questo caso abbiamo bisogno di due elementi fondamentali: l’Adapter (ProductAdapter.java) e il layout della singola riga (product_row_layout.xml).

Il file JSON ricavato dall’interrogazione al DB viene disposto in un JSONArray che ci permette quindi di popolare la lista mediante l’Adapter: scansiono l’array con un ciclo e per ogni passo viene riempita una riga con i quattro elementi detti in precedenza, fin quando non terminano gli elementi degli array.

```
while( count < jsonArray.length() ){
    JSONObject JO = jsonArray.getJSONObject( count );
    nome = JO.getString( "nome" );
    descriz = JO.getString( "descriz" );
    prezzo = JO.getString( "prezzo" );
    Product product = new Product( nome, descriz, prezzo, JO.getString( "id" ), 0 );
    productAdapter.add( product );
    count++;
}
```

Anche in questo caso, utilizziamo un costruttore (Product.java) in modo da gestire più facilmente l'aggiunta della singola riga passando in input i parametri da aggiungere.

Questo file contiene inoltre il metodo setOnClickListener del tasto PRENOTA. Al click del tasto, inizio un ciclo che mi permette di raccogliere e memorizzare il tipo e la quantità di prodotti che voglio prenotare, per poi salvarlo nelle variabili list_obj e

num_obj: creiamo quindi due vettori paralleli in cui in uno è salvato il nome del prodotto e nell'altro, nella medesima posizione, la quantità che si vuole ordinare.

```
prenota.setOnClickListener( new View.OnClickListener() {
    @Override
    public void onClick( View arg0 ) {
        String list_obj = "";
        String num_obj = "";
        for( int i = 0; i < productAdapter.getCount(); ++i ){
            String how_many = ( ( TextView) ( ( LinearLayout ) productAdapter.getView( i, null, null ) ).getChildAt( 3 ) ).getText().toString();
            if( Integer.parseInt( how_many ) > 0 ){
                list_obj += ( ( TextView) ( ( LinearLayout ) productAdapter.getView( i, null, null ) ).getChildAt( 0 ) ).getText().toString();
                num_obj += how_many;
                if( i != productAdapter.getCount()-1 ) {
                    list_obj += ";";
                    num_obj += ";";

                }
            }

        }
```

Successivamente al click del bottone 'Prenota', ci viene richiesto di inserire una propria mail alla quale ricevere la risposta del gestore. In caso di mail scritta correttamente, viene richiamato il costruttore WritingReservationTask dal file WritingReservationTask.java, per poi essere eseguito.

Product.java

Costruttore che ci permette di descrivere un Prodotto tramite:

1) Nome
2) Descrizione
3) Prezzo
4) Id
5) how_many

Quest'ultimo campo è un intero che ci permette di salvare in memoria la quantità di prodotto desiderata dall'utente per poi inviarla al proprietario che deciderà quindi se accettare o meno l'ordine. All'interno del file sono definiti i metodi get e set per ogni variabile.

ProductAdapter.java

Ha gli stessi metodi e gli stessi compiti di ServiceAdapter.java, permettendo quindi (attraverso un Inflater) di convertire il layout della riga (XML) nella groupView che ci consente di raccogliere i dati di interesse. Per salvare i dati definiamo anche qui una classe statica ProductHolder che ha come variabili di tipo TextView :

1) tx_nome
2) tx_descriz
3) tx_prezzo
4) tx_value

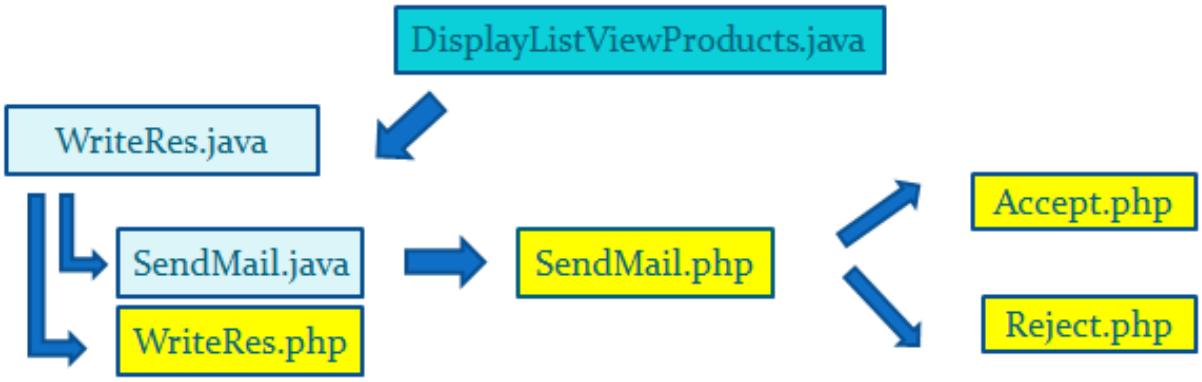
PARTE III – PRENOTAZIONE VIA MAIL

- INVIO RICHIESTA PRENOTAZIONE
- AGGIORNAMENTO TABELLA ‘PRENOTAZIONI’
- RISPOSTA DEL GESTORE

Dopo aver eseguito il click su ‘PRENOTA’, dobbiamo essere in grado di apportare le modifiche necessarie alla tabella ‘Prenotazioni’ nel DB ed inviare una mail al gestore il quale dovrà decidere se accettare o meno la richiesta di prenotazione.

Per fare questo sono stati realizzati i seguenti script:

1) WritingReservationTask.java
2) write_reservation.php
3) sendEmailToAdministrator.java
4) sendEmailToAdministrator.php
5) accept .php
6) reject.php



WritingReservationTask.java

Viene richiamato all’istante della prenotazione ed ha il compito di inserire un record nel tabella “Prenotazioni” del DB remoto con i campi:

1) Timestamp
2) Id del servizio
3) Lista degli oggetti
4) Quantità prenotate
5) Stato della prenotazione (in attesa/accettata/rifiutata)

Per raccogliere questi elementi faccio uso anche qui di un costruttore che definisco all'interno di questo file mostrato in figura:

```
public WritingReservationTask( DisplayListViewProducts act,
                               String id_serv,
                               String lista_ogg,
                               String numero_ogg,
                               String stato,
                               String email
){
    this.act = act;
    this.id_serv = id_serv;
    this.lista_ogg = lista_ogg;
    this.numero_ogg = numero_ogg;
    this.stato = stato;
    this.email = email;
}
```

L'inserimento nel DB avviene tramite lo script *write_reservation.php* richiamato tramite URL da questo file, al quale passo i parametri presi dal costruttore come si può vedere dal codice:

```
timestamp = System.currentTimeMillis();
String json_url = "http://servizieee.altervista.org/php/write_reservation.php?" +
        "timestamp=" + timestamp +
        "&id=" + id_serv +
        "&state=\'" + stato + "\'" +
        "&lista_prod=\'[" + lista_ogg + "]\'" +
        "&num_prod=\'[" + numero_ogg + "]\'" +
        "&email=\'" + email + "\'";
```

Lo script viene infine caratterizzato dal metodo onPostExecute che invia la mail all'Amministratore dei servizi richiamando il file *sendEmailToAdministrator.java* dopo aver mostrato a schermo il messaggio "OK: Reservation sent".

```
@Override
protected void onPostExecute( String result ) {
    Log.d( "DEBUG", result );
    if( result.equalsIgnoreCase( "New record created successfully" ) ) {
        Toast.makeText( act, "OK: Reservation sent", Toast.LENGTH_LONG ).show();
        new sendEmailToAdministrator( act, timestamp, id_serv, email ).execute();
    }
    else
        Toast.makeText( act, "Reservation failed", Toast.LENGTH_LONG ).show();
}
```

write_reservation.php

```php
$timestamp = $_GET["timestamp"];
$id_service = $_GET["id"];
$state_reservation = $_GET["state"];
$lista_prodotti = $_GET["lista_prod"];
$num_prodotti = $_GET["num_prod"];
$email = $_GET["email"];

$sql = "INSERT INTO prenotazioni (timestamp, id_serv, lista_oggetti_pr, numero_oggetti_pr, stato, email) VALUES ($timest
if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
```

Nel momento in cui l'utente preme il tasto "Prenota", viene lanciato il file WritingReservationTask.java, che a sua volta richiama questo script che ha il compito di aggiungere un nuovo record alla tabella 'Prenotazioni'.

timestamp	id_serv	lista_oggetti_pr	numero_oggetti_pr	stato	email
1468486002107	1	[Pizzero;Grillero]	[4;7]	rifiutata	manuel.campo90@gmail.com
1468486176744	1	[Pizzero;Grillero]	[2;3]	rifiutata	manuel.campo90@gmail.com
1468488247954	1	[Pizzero;Grillero]	[1;1]	rifiutata	manuel.campo90@gmail.com
1468488362644	1	[Pizzero;Grillero]	[6;15]	accettata	manuel.campo90@gmail.com
1468488615278	1	[Pizzero;Grillero]	[3;4]	rifiutata	manuel.campo90@gmail.com
1468491013166	3	[Divano;Prive]	[8;3]	accettata	galdieri.roberto@gmail.com
1468491166577	2	[Cuba Libre;Moijto]	[9;6]	in attesa	galdieri.roberto@gmail.com
1468491917584	3	[Divano;Prive]	[5;4]	accettata	manuel.campo90@gmail.com

sendEmailToAdministrator.java

È il file che viene richiamato nel momento in cui la tabella prenotazioni è stata aggiornata regolarmente con un nuovo record. Ha il compito di inviare la mail al gestore.

Il suo costruttore sendEmailToAdministrator riceve in input e tiene in memoria il timestamp della prenotazione, l'id del servizio, la mail al quale inviare la risposta, e la variabile act di tipo DisplayListViewProducts; saranno questi i parametri che, passati nella URL, consentiranno l'invio della mail all'Amministratore tramite il file *sendEmailToAdministrator.php*

sendEmailToAdministrator.php

Richiamato dal file sendEmailToAdministrator.java, questo script memorizza con metodo GET i parametri passati dalla URL 'id_serv' , 'timestamp' ed 'email_rec'.

```php
$id_serv = $_GET['id_serv'];
$timestamp = $_GET['timestamp'];
$email_rec = $_GET['email_rec'];
$list_obj = $_GET['list_obj'];
$num_obj = $_GET['num_obj'];
//$email_rec = 'galdieri.roberto@gmail.com';

$to      = 'manuel.campo90@gmail.com';
$subject = 'Nuova prenotazione da ' . $email_rec . ': ACCETTI O RIFIUTI ?';
//$subject = 'Nuova prenotazione da ' . 'galdieri.roberto@gmail.com' . ': ACCETTI O RIFIUTI ?';
$message = "lista degli oggetti " . $list_obj . " numero degli oggetti " . $num_obj . "\r\n" . "\r\n";
$message = $message . 'ACCETTO: ' . 'http://servizieee.altervista.org/php/accept.php?timestamp=' . $timestamp . '&id_serv=' . $id_serv
$message = $message . "\r\n" . "\r\n";
$message = $message . 'RIFIUTO: http://servizieee.altervista.org/php/reject.php?timestamp=' . $timestamp . '&id_serv=' . $id_serv . '&
$headers = 'From: ' . $email_rec . "\r\n" .
    'Reply-To: ' . $email_rec . "\r\n" .
    'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);
```

Supponiamo inoltre che l'indirizzo di posta elettronica dell'amministratore sia assegnato staticamente (in questo caso, manuel.campo90@gmail.com): quest'ultimo riceverà una richiesta di prenotazione in cui leggerà l'indirizzo mail del richiedente e deciderà se accettare o meno la prenotazione tramite link.

I link fanno riferimento ad altri due file php:

1) *Accept.php*
2) *Reject.php*

accept .php

Nel caso in cui la richiesta venga accettata, il richiedente riceverà una mail di conferma con l'id relativo alla prenotazione e il timestamp del momento in cui è stata accettata.

```php
$sql = "UPDATE prenotazioni SET stato='accettata' WHERE id_serv=$id_serv and timestamp=$timestamp";
if ($conn->query($sql) === TRUE) {
    echo $sql . "   ->   Executed <br/>";
    $email_rec = str_replace("'", "", $email_rec);
    $to      = $email_rec;
    $subject = 'Prenotazione accettata!';
    $message = 'La tua prenotazione con id ' . $id_serv . ' e timestamp ' . $timestamp . 'è stata accettata.';
    $headers = 'From: OurApplication@bo.com' . "\r\n" .
        'Reply-To: OurApplication@bo.com' . "\r\n" .
        'X-Mailer: PHP/' . phpversion();
    mail($to, $subject, $message, $headers);

    echo "EMAIL SENT: " . $id_serv ." ". $timestamp ." ". $email_rec;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
```

Inoltre, nella tabella ‘Prenotazioni’ viene settato lo stato della prenotazione come “ACCETTATA” in corrispondenza del record con quel determinato timestamp.

reject.php

Nel caso in cui la richiesta venga rifiutata, il richiedente riceverà una mail con l’id relativo alla prenotazione e il timestamp del momento in cui è stata rifiutata.

```php
$sql = "UPDATE prenotazioni SET stato='rifiutata' WHERE id_serv=$id_serv and timestamp=$timestamp";
if ($conn->query($sql) === TRUE) {
    echo $sql . "    ->    Executed <br/>";
    $email_rec = str_replace("'", "", $email_rec);
    $to      = $email_rec;
    $subject = 'Prenotazione rifiutata!';
    $message = 'La tua prenotazione con id ' . $id_serv . ' e timestamp ' . $timestamp . 'è stata rifiutata.';
    $headers = 'From: OurApplication@bo.com' . "\r\n" .
        'Reply-To: OurApplication@bo.com' . "\r\n" .
        'X-Mailer: PHP/' . phpversion();

    mail($to, $subject, $message, $headers);

    echo "EMAIL SENT: " . $id_serv ." ". $timestamp ." ". $email_rec;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Inoltre, nella tabella ‘Prenotazioni’ viene settato lo stato della prenotazione come “RIFIUTATA” in corrispondenza del record con quel determinato timestamp.

Se la prenotazione non è stata né accettata, né rifiutata, quest’ultima è considerata in stato di “ATTESA”.

CONCLUSIONI

Spinti dalla curiosità per un mondo, quello Android, a noi completamente sconosciuto, abbiamo creato una app la cui funzione è quella di prenotare alcuni servizi offerti da un locale, per esempio appunto la ristorazione, la prenotazione tavoli, ..etc. Alla fine del lavoro ci siamo accorti come il lavoro da noi svolto presenti una struttura ben organizzata e potrebbe quindi essere riadattata in diversi contesti per scopi simili.