SMART LOYALTY

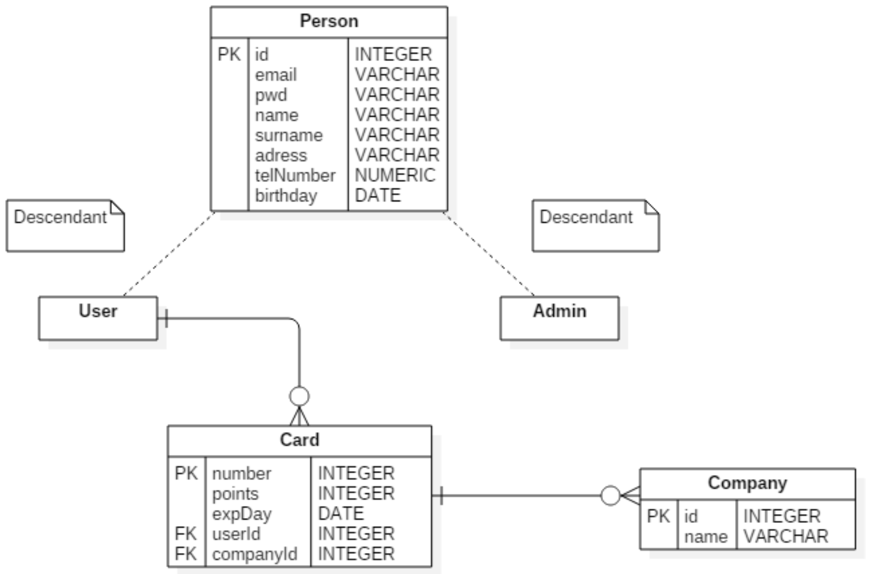
SUMMARY

1. DATABASE

1.1. Structure

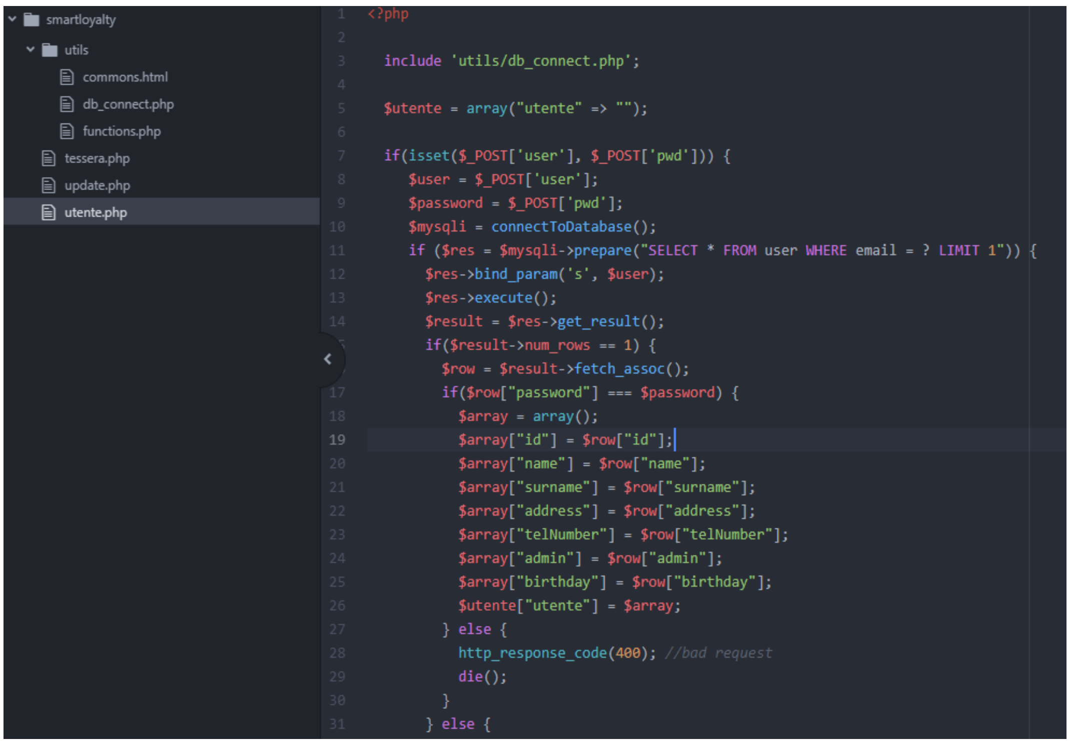
The following structure is only related the user part.

There are respectively three entities the Person entity, that is the generalization of admin and user, the Card and the relative Company of appurtenance.

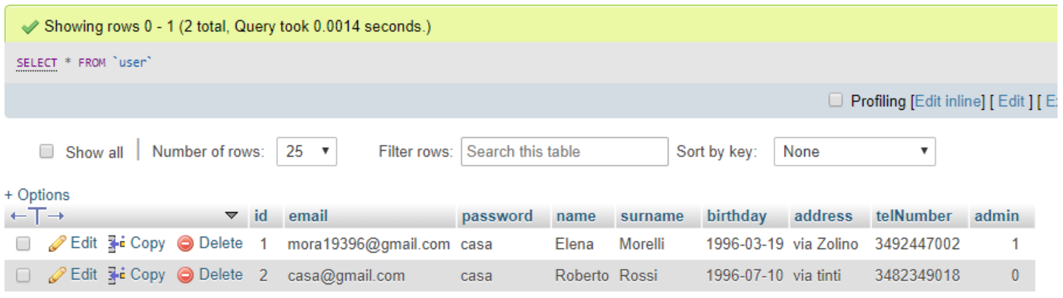


1.2. Technology

The database was created in an Apache sever using the mysql language. To connect to it php pages are used.



```php
<?php

  include 'utils/db_connect.php';

  $utente = array("utente" => "");

  if(isset($_POST['user'], $_POST['pwd'])) {
     $user = $_POST['user'];
     $password = $_POST['pwd'];
     $mysqli = connectToDatabase();
     if ($res = $mysqli->prepare("SELECT * FROM user WHERE email = ? LIMIT 1")) {
       $res->bind_param('s', $user);
       $res->execute();
       $result = $res->get_result();
       if($result->num_rows == 1) {
         $row = $result->fetch_assoc();
         if($row["password"] === $password) {
           $array = array();
           $array["id"] = $row["id"];
           $array["name"] = $row["name"];
           $array["surname"] = $row["surname"];
           $array["address"] = $row["address"];
           $array["telNumber"] = $row["telNumber"];
           $array["admin"] = $row["admin"];
           $array["birthday"] = $row["birthday"];
           $utente["utente"] = $array;
         } else {
           http_response_code(400); //bad request
           die();
         }
       } else {
```


Showing rows 0 - 1 (2 total, Query took 0.0014 seconds.)

```
SELECT * FROM `user`
```

Profiling [Edit inline] [Edit] [E
Show all | Number of rows: 25
Filter rows: Search this table
Sort by key: None
+ Options

	id	email	password	name	surname	birthday	address	telNumber	admin
Edit Copy Delete	1	mora19396@gmail.com	casa	Elena	Morelli	1996-03-19	via Zolino	3492447002	1
Edit Copy Delete	2	casa@gmail.com	casa	Roberto	Rossi	1996-07-10	via tinti	3482349018	0

2. CLASSES

2.1. Activity

2.1.1. Main Activity

It has the task to manage all the fragments in the Navigation drawer and add the user in the Room Database in case is not already there.

```
/**
 * Method to get the new fragment to add to the screen
 * @param state the current state from which will be recovered the
   fragment
 * @return a pair containing the fragment and a tag to be assigned
 */
private Pair<Fragment, String> getFragment(final MenuState state) {
    Fragment fr;
    String tag;
    switch (state){
        case HOME:
            fr = HomeFragment.newInstance();
            tag = MenuState.HOME.toString();
            break;
        case CARD:
            fr = CardListFragment.newInstance();
            tag = MenuState.CARD.toString();
            break;
        case PROFILE:
            fr = ProfileFragment.newInstance();
            tag = MenuState.PROFILE.toString();
            break;
        default:
            fr = EmptyFragment.newInstance();
            tag = "empty";
            break;
    }
    return Pair.create(fr, tag);
}
```

```
@Override
public void onLoginCompleted(final User user, final boolean newUser) {
    if(newUser){
        this.manager.addUser(user);
    } else {
        this.manager.setUser(user);
    }
}
```

2.1.2 LogIn and Logged Activity

The LogIn activity has the task to check if the email is in the database and if the password related to that email is correct. To do this the class has an AsyncTask that check Online in case there is internet connection otherwise in the room database.

```
/**
     * Represents an asynchronous login/registration task used to
authenticate
     * the user.
     */
    /**
     * Represents an asynchronous login/registration task used to
authenticate
     * the user.
     */
    private class UserLoginTask extends AsyncTask<Void, Void, Integer> {

        private static final String USER_FIELD = "utente";

        private final String mEmail;
        private final String mPassword;
        private int admin = 0;
        private User user;
        private boolean addUser = false;

        UserLoginTask(String email, String password) {
            mEmail = email;
            mPassword = password;
        }

        @Override
        protected Integer doInBackground(Void... params) {

            UserDAO userTable =
AppDatabase.getDatabase(LoginActivity.this).userDao();
            User userDb = userTable.getUser(mEmail);
            final boolean isOnline =
ConnectionUtilities.isConnectionAvailable(LoginActivity.this);
            if(userDb == null || (userDb.hasToUpdate() && isOnline)){
                if(isOnline) {
                    final Map<String, String> map =
ConnectionUtilities.getLoginMap(mEmail, mPassword);
                    final String url = ConnectionUtilities.SERVER_HTTP_URL
+ ConnectionUtilities.LOGIN_PAGE;
                    try {
                        final String response =
ConnectionUtilities.getDataFromUrl(url, map, true);
                        JSONObject userJS = new
JSONObject(response).getJSONObject(USER_FIELD);
                        if (userJS != null) {
                            userJS.put(User.EMAIL_FIELD, mEmail);
                            userJS.put(User.PWD_FIELD, mPassword);
                            userJS.put(User.ADMIN_FIELD,admin);
                            this.user = new User(userJS);
                            this.addUser = userDb == null;
                            return HttpURLConnection.HTTP_OK;
                        }
                    } catch (HttpException e) {
                        return e.getCode();
                    } catch (JSONException e) {
                        Log.e(TAG, e.toString());
                    }
                } else {
                    return HttpURLConnection.HTTP_GONE;
                }
            } else {
                this.user = userDb;
                if(this.user.getPwd().equals(mPassword)){
                    return HttpURLConnection.HTTP_OK;
                } else {
                    return HttpURLConnection.HTTP_BAD_REQUEST;
                }
            }

            return -2;
        }

        @Override
        protected void onPostExecute(final Integer result) {
            mAuthTask = null;

            switch (result) {
                // the password is incorrect
                case HttpURLConnection.HTTP_BAD_REQUEST:

mPasswordView.setError(getString(R.string.error_incorrect_password));
                    mPasswordView.requestFocus();
                    break;
                // everything is good, login successful
                case HttpURLConnection.HTTP_OK:
                    SharedPreferences pref =
LoginActivity.this.getPreferences(Context.MODE_PRIVATE);
                    SharedPreferences.Editor editor = pref.edit();
                    editor.putString(USER_NAME, this.mEmail);
                    editor.apply();
                    Intent returnIntent = new Intent();
                    returnIntent.putExtra(USER_DATA, this.user);
                    returnIntent.putExtra(USER_CREATE, this.addUser);
                    setResult(Activity.RESULT_OK, returnIntent);
                    finish();
                    break;
                // the user doesn't exist
                case HttpURLConnection.HTTP_NOT_FOUND:

mEmailView.setError(getString(R.string.error_incorrect_email));
                    mEmailView.requestFocus();
                    break;
                case HttpURLConnection.HTTP_GONE:

ConnectionUtilities.alertConnectionAbsence(LoginActivity.this);
                    break;
                default:
ConnectionUtilities.alertConnectionError(LoginActivity.this);
                    break;

            }
        }

        @Override
        protected void onCancelled() {
            mAuthTask = null;
        }

    }
}
```

The LoggedIn class is abstact and gets the result from the Login activity and through the abstRact method onLogInCompleted get the authenticated user to the MainActivity.

```
/**
 * Abstract method to inform the subclasses that the login is completed
 * @param user the user that logged
 * @param newUser if the user it's using this app for the first time
 */
public abstract void onLoginCompleted(final User user, final boolean
newUser);

private void login() {
    Intent intent = new Intent(this,LoginActivity.class);
    startActivityForResult(intent, LOG_REQ_CODE);
}
```

2.2 Database

The database has two entities for now: the User the Card with relative Dao classes

```
@Dao
public interface UserDAO {
    @Query("SELECT * FROM user WHERE email LIKE :email")
    LiveData<User> getObservableUser(final String email);

    @Query("SELECT * FROM user WHERE email LIKE :email")
    User getUser(final String email);

    @Update
    void updateUser(final User user);

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void addUser(final User user);

}
```

```
@Dao
public interface CardDAO {

    @Query("SELECT * FROM card WHERE cardId = :id")
    LiveData<Card> getCardFromId(final int id);

    @Update
    void updateCard(final Card card);

    @Query("SELECT * FROM card WHERE user_email LIKE :email ")
    List<Card> getCardsFromUserEmail(final String email);

    @Query("SELECT * FROM card WHERE cardId = :id")
    LiveData<Card> getObservableCardFromUserEmail(final int id);

    @Query("SELECT * FROM card WHERE user_email LIKE :email ")
    LiveData<List<Card>> getObservableCardsFromUserEmail(final String
email);

    @Insert
    void insertCard(final Card card);
}
```

2.3 Fragments

There are 4 different fragments for now: the Profile fragment, the Changeinfo fragment, CardList fragment and the Card fragment.

The first one has the user profile data while the second one enable to change them like the password for example.

The CardList shows every card that the user has while the Card Fragment all the data from the one selected in the list.

```
public class CardListFragment extends Fragment {

    private InformationManager manager;
    private String userEmail;
    private static final String TAG = "Card_List";
    private Context context;
    private CardAdapter cardAdapter;

    public static CardListFragment newInstance() {
        return new CardListFragment();
    }

    public CardListFragment(){}

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.manager =
ViewModelProviders.of(getActivity()).get(InformationManager.class);
        new edu.ktu.smart_loyalty.utils.ParallelExecutor().execute(new
Runnable() {
            @Override
            public void run() {
                userEmail = manager.getUser().getEmail();
            }
        });
        this.context = getContext();
    }

    @Override
    public View onCreateView(final LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        final View view = inflater.inflate(R.layout.card_list, container,
false);
        final ListView layout = (ListView)
view.findViewById(R.id.cardListView);
        final LiveData<List<Card>> cardList =
this.manager.getUserCards(userEmail);
        final List<Card> list = new ArrayList<>();
        this.cardAdapter = new CardAdapter(
                getActivity(),
                android.R.layout.simple_list_item_1,
                list
        );
        layout.setAdapter(cardAdapter);

        final MainActivity main = (MainActivity) getActivity();

        cardList.observe(this, new Observer<List<Card>>() {
            @Override
            public void onChanged(@Nullable List<Card> cards) {
                if (cards != null) {
                    if (cards.isEmpty()) {
                        main.noDataAvailable();
                    } else {
                        list.clear();
                        list.addAll(cards);
                        cardAdapter.notifyDataSetChanged();
                    }
                }
            }
        });

        layout.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {

               Fragment nextFrag=
CardFragment.newInstance(list.get(position));

getActivity().getSupportFragmentManager().beginTransaction()
                        .replace(R.id.fragment_layout,
nextFrag,"findThisFragment")
                        .addToBackStack(null)
                        .commit();
            }
        });

        return view;
    }

    @Override
    public void onResume() {
        super.onResume();
        if(!ConnectionUtilities.isConnectionAvailable(getContext())){
            Toast.makeText(getContext(), getString(R.string.warning_text),
Toast.LENGTH_LONG).show();
        }
    }
```

```
public class ChangeInfoFragment extends Fragment implements
UserRepository.UpdateListener {

    public static ChangeInfoFragment newInstance() {
        return new ChangeInfoFragment();
    }

    private ChangeInfoFragmentListener mListener;
    private InformationManager manager;
    private User user;
    private TextView name;
    private TextView date;
    private ImageView profileImage;
    private EditText addr;
    private EditText num;
    private EditText mail;
    private EditText pwd;

    public ChangeInfoFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        final View view = inflater.inflate(R.layout.fragment_change_info,
container, false);
        this.manager =
ViewModelProviders.of(getActivity()).get(InformationManager.class);

        this.name = view.findViewById(R.id.change_user_name);
        this.date = view.findViewById(R.id.change_user_date);
        this.profileImage = view.findViewById(R.id.change_user_img);
        this.addr = view.findViewById(R.id.change_user_addr);
        this.num = view.findViewById(R.id.change_user_num);
        this.mail = view.findViewById(R.id.change_user_email);
        this.pwd = view.findViewById(R.id.change_user_pwd);

        final Button btn = view.findViewById(R.id.change_complete);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mListener != null) {
                    final String addrText = addr.getText().toString();
                    final String numText = num.getText().toString();
                    final String mailText = mail.getText().toString();
                    final String pwdText = pwd.getText().toString();
                    if(addrText.isEmpty() || numText.isEmpty() ||
mailText.isEmpty() || pwdText.isEmpty()) {
                        Toast.makeText(getContext(), R.string.empty_fields,
Toast.LENGTH_SHORT).show();
                    } else {
                        user.setAddress(addrText);
                        user.setNumber(numText);
                        user.setEmail(mailText);
                        user.setPwd(pwdText);
                        manager.updateUser(user, ChangeInfoFragment.this);
                    }
                }
            }
        });
        new SetUserFields().execute();
        return view;
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof ChangeInfoFragmentListener) {
            mListener = (ChangeInfoFragmentListener) context;
        } else {
            throw new RuntimeException(context.toString()
                    + " must implement OnFragmentInteractionListener");
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
    }

    @Override
    public void onUpdateComplete(boolean isUpdated) {
        if(isUpdated){
            mListener.onChangesCompleted();
        } else {
            if(ConnectionUtilities.isConnectionAvailable(getContext())){
                ConnectionUtilities.alertConnectionError(getContext());
            } else {
                ConnectionUtilities.alertConnectionAbsence(getContext());
            }
        }
    }

    public interface ChangeInfoFragmentListener {
        void onChangesCompleted();
    }

    private class SetUserFields extends AsyncTask<Void, Void, User>{
        @Override
        protected User doInBackground(Void... voids) {
            return manager.getUser();
        }

        @Override
        protected void onPostExecute(User newUser) {
            user = newUser;
            SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMMM
YYYY", Locale.getDefault());
            name.setText(newUser.getFullName());
            date.setText(dateFormat.format(newUser.getBirthDate()));
            addr.setText(newUser.getAddress());
            num.setText(newUser.getNumber());
            mail.setText(newUser.getEmail());
            pwd.setText(newUser.getPwd());
        }
    }
```

2.4 Repository

In this package the is the UserRepository that manages all the task that involve the data got from the database

```
public class UserRepository {
    private static final String TAG = "user_repo_tag";

    private final Context context;
    private UserDAO userTable;
    private CardDAO cardTable;
    private UserInformationListener listener;
    private LiveData<User> user;
    private LiveData<List<Card>> card;
    private String email;

    private ParallelExecutor executor = new ParallelExecutor();

    public interface UserInformationListener {
        void onCardUpdate(Card newCard);
        void onListCardUpdate(List<Card> card);
    }

    public interface UpdateListener{
        void onUpdateComplete(boolean isUpdated);
    }

    public UserRepository(final Context context){
        this.context = context;
        AppDatabase db = AppDatabase.getDatabase(context);
        userTable = db.userDao();
        cardTable = db.cardDAO();

    }

    public LiveData<User> getObservableUser(){
        return this.user;
    }

    public User getUser(){
        return userTable.getUser(email);
    }

    public void setUser(final User user){
        executor.execute(new Runnable() {
            @Override
            public void run() {
                User dbUser = userTable.getUser(user.getEmail());
                if(dbUser != null && !dbUser.equals(user)){
                    userTable.updateUser(user);
                }
            }
        });
        email = user.getEmail();
        this.user = userTable.getObservableUser(email);
        this.card = cardTable.getObservableCardsFromUserEmail(email);
    }

    public void updateUser(final User newUser, final UpdateListener
listener){
        if(newUser.isUpdated(this.user.getValue())){
            new UserUpdateTask(newUser, listener).execute();
        } else {
            listener.onUpdateComplete(true);
        }
    }

    public void addUser(final User newUser){
        executor.execute(new Runnable() {
            @Override
            public void run() {
                String user_email = newUser.getEmail();
                final String url_card = ConnectionUtilities.SERVER_HTTP_URL
+ ConnectionUtilities.CARD_PAGE;
                try {
                    Map<String, String> map =
ConnectionUtilities.getCardMap(user_email);
                    String data =
ConnectionUtilities.getDataFromUrl(url_card, map, true);
                    List<Card> cardList = new ArrayList<>();
                    if (data != null && !data.isEmpty()) {
                        JSONArray jsonArray = new JSONArray(data);
                        for (int i = 0; i < jsonArray.length(); i++) {
                            JSONObject objData =
jsonArray.optJSONObject(i);
                            final String number =
objData.getString("number");
                            final String expiration =
objData.getString("expDay");
                            final String points =
objData.getString("points");
                            final String idCompany =
objData.getString("codCompany");
                            final String name =
objData.getString("nameCompany");
                            final String lastUpdate =
Card.FORMATTER.format(Calendar.getInstance().getTime());
                            Card card = new
Card(number,name,expiration,Integer.valueOf(points),Integer.valueOf(idCompa
ny),user_email,lastUpdate);
                            cardTable.insertCard(card);
                        }
                        userTable.addUser(newUser);
                        email = user_email;
                        user = userTable.getObservableUser(user_email);
                        card =
cardTable.getObservableCardsFromUserEmail(user_email);
                    }
                } catch (HttpException e) {
                    Log.e(TAG, "Error in userRepository addUser, HttpCode:
" + e.getCode(), e);
                } catch (JSONException e) {
                    Log.e(TAG, "Error in userRepository addUser", e);
                }
            }
        });


    }

    public LiveData<List<Card>> getUserCards(final String userEmail){
       /* executor.execute(new Runnable() {
            @Override
            public void run() {
                List<Card> list =
cardTable.getCardsFromUserEmail(userEmail);
                if(list != null && !list.isEmpty()){
                    new CardCheckUpdate();
                }
            }
        });*/
        email = userEmail;
        this.card = cardTable.getObservableCardsFromUserEmail(email);
        return card;
    }
private class UserUpdateTask extends AsyncTask<Void, Void, Boolean> {

    private final User newUser;
    private final UpdateListener listener;

    public UserUpdateTask(final User newUser, final UpdateListener
listener){
        this.newUser = newUser;
        this.listener = listener;
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        final String oldEmail = new String(email);
        final String newEmail = newUser.getEmail();
        final Map<String, String> map =
ConnectionUtilities.getUserUpdateMap(
                this.newUser.getName(),
                this.newUser.getSurname(),
                this.newUser.getAddress(),
                this.newUser.getNumber(),
                this.newUser.getEmail(),
                this.newUser.getPwd()
        );
        final String url = ConnectionUtilities.SERVER_HTTP_URL +
ConnectionUtilities.UPDATE_PAGE;
        try {
            if(ConnectionUtilities.isConnectionAvailable(context)) {
                ConnectionUtilities.getDataFromUrl(url, map, true);
                userTable.updateUser(newUser);
                if(!oldEmail.equals(newEmail)) {
                    email = newEmail;
                   List<Card> userCard =
cardTable.getCardsFromUserEmail(oldEmail);
                    for (Card c:userCard) {
                        c.setUserEmail(newEmail);
                    }
                    user = userTable.getObservableUser(newUser.getEmail());
                    card =
cardTable.getObservableCardsFromUserEmail(email);
                }
                return true;
            }
        } catch (Exception e) {
            Log.e(TAG, "error in userRepository update info", e);
            return false;
        }
        return false;
    }



    @Override
    protected void onPostExecute(final Boolean success) {
        if (!success) {
            if(this.listener != null){
                this.listener.onUpdateComplete(false);
            }
        } else {
            if(this.listener != null){
                this.listener.onUpdateComplete(true);
            }
        }
    }
}
```

2.5 Connection

Contains all that method that make possible for the app to connect to the php code

```
               public class ConnectionUtilities {
    /* server url */
    private static final String IP_ADDRESS = "192.168.137.1:80";
    public static final String SERVER_HTTP_URL = "http://" + IP_ADDRESS +
"/smartloyalty/";

    /* web pages */
    public static final String LOGIN_PAGE = "utente.php";
    public static final String CARD_PAGE = "tessera.php";
    public static final String UPDATE_PAGE = "update.php";


    /* data requested by the login page */
    private static final String LOGIN_USER = "user";
    private static final String LOGIN_PWD = "pwd";

    /* data requested by the update user info page */
    private static final String UPDATE_NAME = "nome";
    private static final String UPDATE_SURNAME = "cognome";
    private static final String UPDATE_ADDR = "ind";
    private static final String UPDATE_PHONE_NUMBER = "tel";
    private static final String UPDATE_EMAIL = "email";
    private static final String UPDATE_PWD = "pwd";

    /* data requested by the card page */
    private static final String CARD_USER = "email";


    private ConnectionUtilities() {
    }

    /**
     * Creates a dialog that informs about an error
     *
     * @param context
     */
    public static void alertConnectionError(final Context context) {
        new AlertDialog.Builder(context)
```

```java
                .setTitle(R.string.error_sending_data_title)
                .setMessage(R.string.error_sending_data)
                .setPositiveButton("ok", null)
                .show();
    }

    /**
     * Creates a dialog that informs about the connection absence
     *
     * @param context
     */
    public static void alertConnectionAbsence(final Context context) {
        new AlertDialog.Builder(context)
                .setTitle(R.string.connection_absence_title)
                .setMessage(R.string.connection_absence_data)
                .setPositiveButton("ok", null)
                .show();
    }

    /**
     * Retrieve data from an url
     *
     * @param urlToConnect the url to connect
     * @param params       post params to send
     * @param isPost       if it's a post or a get
     * @return a string with the content of the response
     * @throws HttpException if a http error occurs
     */
    public static String getDataFromUrl(final String urlToConnect, final
Map<String, String> params, final boolean isPost) throws HttpException {
        HttpURLConnection httpURLConnection = null;
        StringBuilder response = new StringBuilder();
        BufferedReader rd = null;

        try {
            URL url = new URL(urlToConnect);
            httpURLConnection = (HttpURLConnection) url.openConnection();
            httpURLConnection.setConnectTimeout(3 * 1000);
            httpURLConnection.setReadTimeout(20 * 1000);
            httpURLConnection.setUseCaches(false);
            httpURLConnection.setRequestMethod(isPost ? "POST" : "GET");
            httpURLConnection.setDoInput(true);

            if (isPost) {
                httpURLConnection.setDoOutput(true);
                httpURLConnection.setChunkedStreamingMode(0);
                OutputStream os = httpURLConnection.getOutputStream();
                BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(os, "UTF-8"));
                writer.write(getPostDataString(params));
                writer.flush();
                writer.close();
                os.close();
            }

            int responseCode = httpURLConnection.getResponseCode();
            switch (responseCode) {
                case HttpURLConnection.HTTP_OK:
                    InputStream inputStream =
httpURLConnection.getInputStream();
                    rd = new BufferedReader(new
InputStreamReader(inputStream));
                    String line = "";
                    while ((line = rd.readLine()) != null) {
                        response.append(line);
                    }
                    break;
                default:
                    throw new HttpException(responseCode);
            }
        } catch (IOException e) {
            e.printStackTrace();
            if (e instanceof SocketTimeoutException) {
                throw new HttpTimeoutException();
            }
        } finally {
            if (rd != null) {
                try {
                    rd.close();
                } catch (Exception e) {
                }
            }
            if (httpURLConnection != null) {
                httpURLConnection.disconnect();
            }
        }
        return response.toString();
    }

    public static boolean isConnectionAvailable(final Context context) {
        ConnectivityManager connectivityManager = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo =
connectivityManager.getActiveNetworkInfo();
        if (networkInfo == null || !networkInfo.isConnected() ||
                (networkInfo.getType() != ConnectivityManager.TYPE_WIFI &&
networkInfo.getType() != ConnectivityManager.TYPE_MOBILE)) {
            return false;
        }
        return true;
    }

    /**
     * Get a string with the post parameters
     *
     * @param params a map with the data to send
     * @return
     * @throws UnsupportedEncodingException
     */
    public static String getPostDataString(final Map<String, String>
params) throws UnsupportedEncodingException {
        StringBuilder res = new StringBuilder();
        boolean first = true;
        for (Map.Entry<String, String> entry : params.entrySet()) {
            if (first) {
                first = false;
            } else {
                res.append("&");
            }
            res.append(URLEncoder.encode(entry.getKey(), "UTF-8"));
            res.append("=");
            res.append(URLEncoder.encode(entry.getValue(), "UTF-8"));
        }
        return res.toString();
    }

    /**
     * Creates a map with the post data to send to the login page
     *
     * @param user     the user email
     * @param password the user password
     * @return a map with the data
     */
    public static Map<String, String> getLoginMap(final String user, final
String password) {
        HashMap<String, String> map = new HashMap<>();
        map.put(LOGIN_USER, user);
        map.put(LOGIN_PWD, password);
        return map;
    }

    /**
     * Creates a map with the post data to send to the user update page
     * @param name user's name
     * @param surname user's surname
     * @param add user's address
     * @param phone user's phone number
     * @param mail user's email
     * @param pwd user password
     * @return a map with the data
     */
    public static Map<String, String> getUserUpdateMap(final String name,
final String surname, final String add, final String phone, final String
mail, final String pwd) {
        HashMap<String, String> map = new HashMap<>();
        map.put(UPDATE_NAME, name);
        map.put(UPDATE_SURNAME, surname);
        map.put(UPDATE_EMAIL, mail);
        map.put(UPDATE_PHONE_NUMBER, phone);
        map.put(UPDATE_ADDR, add);
        map.put(UPDATE_PWD, pwd);
        return map;
    }

    /**
     * Creates a map with the post data to send to the card page
     * @param user the user's email
     * @return a map with the data
     */
    public static Map<String, String> getCardMap(final String user) {
        HashMap<String, String> map = new HashMap<>();
        map.put(CARD_USER, user);
        return map;
    }
```

2.6 Information Manager

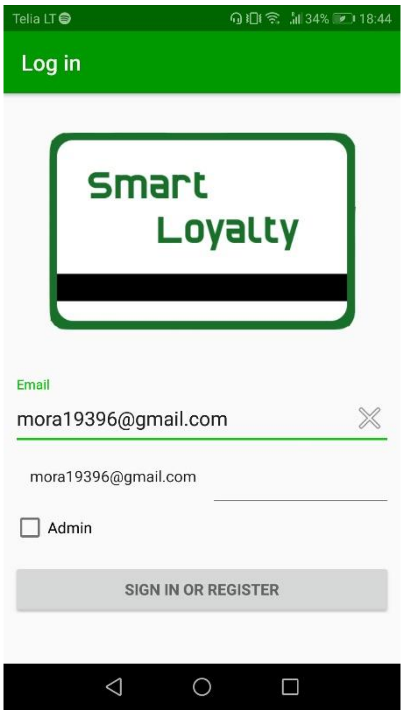
The information Manager Class has the task to get all the result from the repository classes

```
public class InformationManager extends AndroidViewModel {
    private UserRepository userRepo;
    private Context context;

    /**
```

```java
     * Constructor.
     *
     * @param application
     */
    public InformationManager(final Application application) {
        super(application);
        this.context = application;
        this.userRepo = new UserRepository(application);
    }

    public LiveData<User> getObservableUser(){
        return this.userRepo.getObservableUser();
    }

    public User getUser() {
        return this.userRepo.getUser();
    }

    public LiveData<List<Card>> getUserCards(final String userEmail){
        return userRepo.getUserCards(userEmail);
    }

    public void updateUser(final User user, final
UserRepository.UpdateListener listener){
        this.userRepo.updateUser(user, listener);
    }

   public void addUser(final User user){
        this.userRepo.addUser(user);
    }

    public void setUser(final User user){
        this.userRepo.setUser(user);
    }
```

3. SCREENSHOTS

3.1 Log In


Telia LT 34% 18:44

Log in

Smart Loyalty

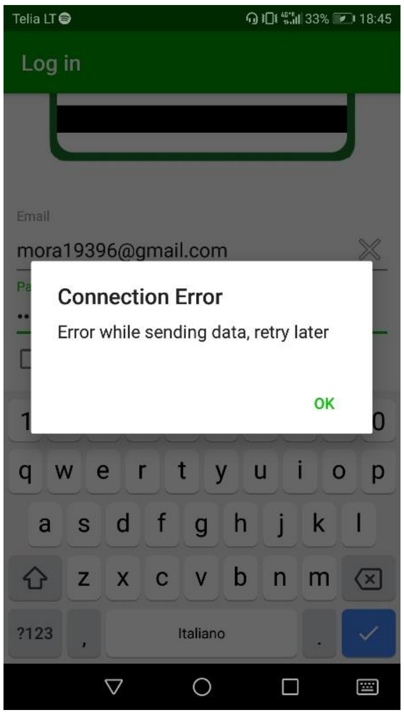
Email

mora19396@gmail.com

mora19396@gmail.com

Admin

SIGN IN OR REGISTER


Telia LT 33% 18:45

Log in

Email

mora19396@gmail.com

Connection Error

Error while sending data, retry later

OK

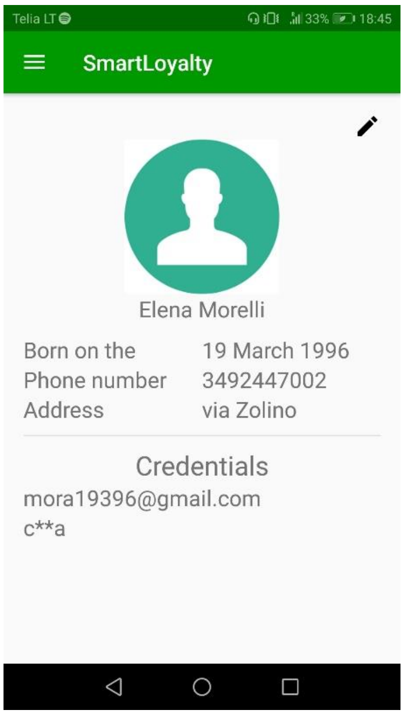
Log In screen

In the second screen there is an Info Dialog, in case there is going to be some problem with the connection .

The Admin checkbox will be able to redirect the user in the admin section if he is one.

All the email used will be saved and the email field has the auto complete.

3.2 Profile


Telia LT 33% 18:45

SmartLoyalty

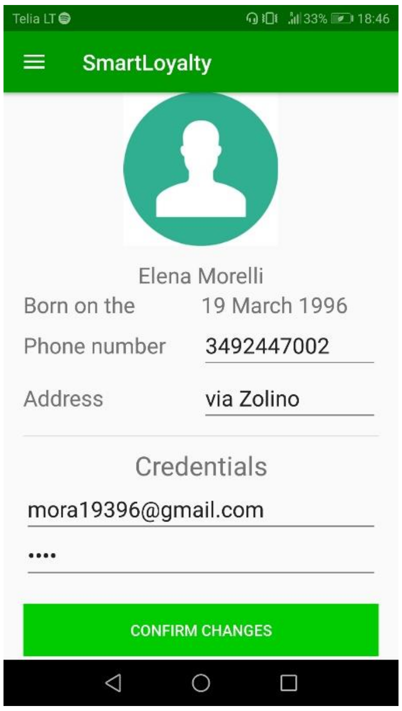
Elena Morelli

Born on the	19 March 1996
Phone number	3492447002
Address	via Zolino

Credentials

mora19396@gmail.com

c**a


Telia LT 33% 18:46

SmartLoyalty

Elena Morelli

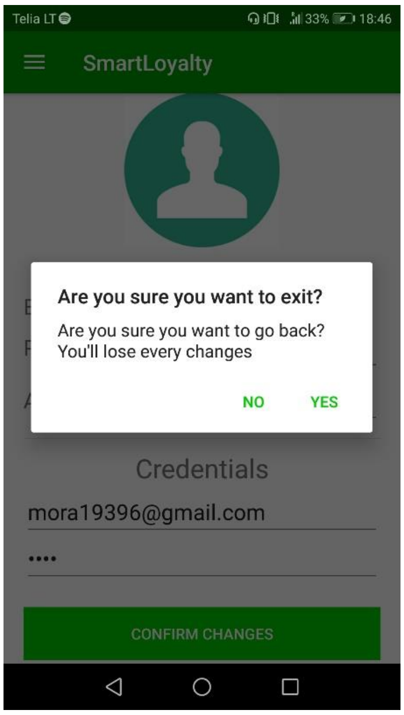
Born on the	19 March 1996
Phone number	3492447002
Address	via Zolino

Credentials

mora19396@gmail.com

••••

CONFIRM CHANGES


Telia LT 33% 18:46

SmartLoyalty

Are you sure you want to exit?

Are you sure you want to go back? You'll lose every changes

NO YES

Credentials

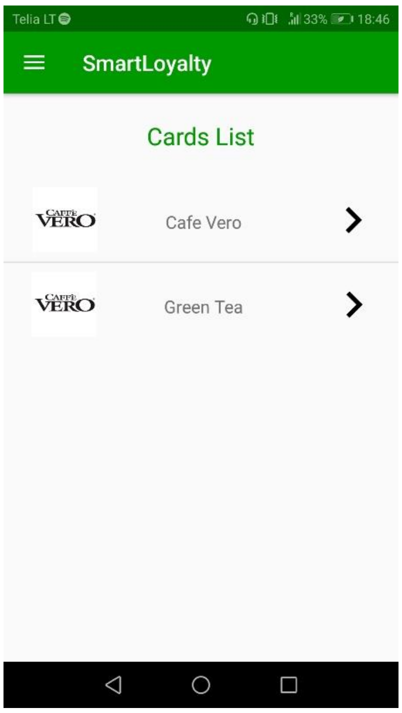
mora19396@gmail.com

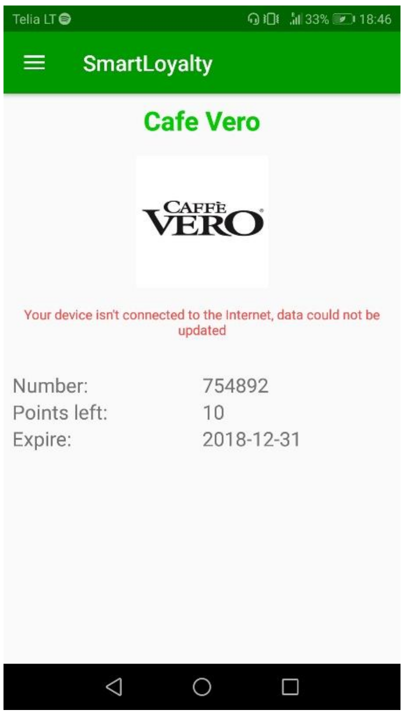
••••

CONFIRM CHANGES

The first figure is the user profile while the second is the fragment that you get when you click the edit button. The third screen is the info Dialog that the user will get if he press the back button before confirming the changes.

3.3 Card List and Card Viewer





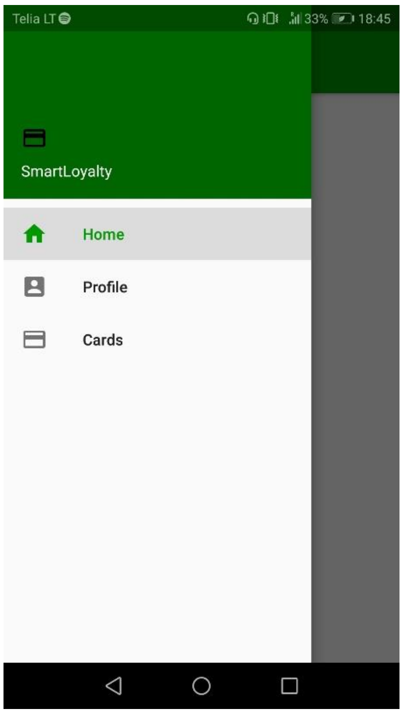
Your device isn't connected to the Internet, data could not be updated

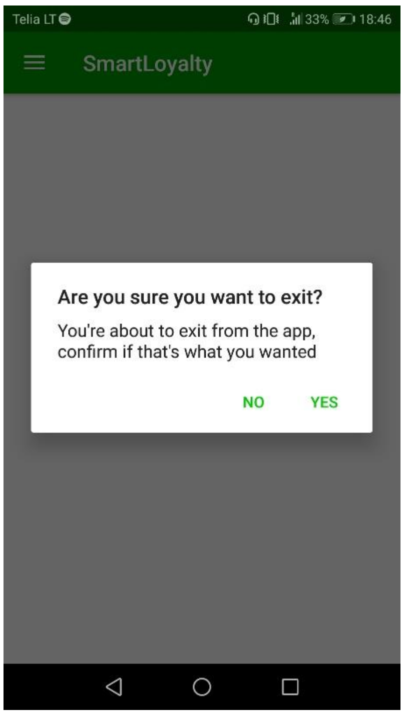
Number: 754892
Points left: 10
Expire: 2018-12-31

Fragment with list of all the cards.

The fragment that shows the selected card.

3.4 Menu and Exit Dialog





Are you sure you want to exit?

You're about to exit from the app, confirm if that's what you wanted

NO YES

Drawer Menu and the info Dialog when the user is about to leave the app.