**ВСТУП**

**Актуальність теми:** полягає в тому, щоб дозволити користувачам обмінюватись фотографіями, тобто вести свій блог, та особам які ведуть підприємницьку діяльність могли демонструвати свої товари, з метою збільшення фінансового обороту їхнього бізнесу. Для цього створена ця програма і для неї використовується Backend з використанням Parse.

В даній курсовій роботі буде розроблений додаток на базі операційної системи Android для роботи з backend, який буде надавати змогу користувачеві переглядати свої так і інших користувачі медіа файли, можливість реєстрації/авторизації, написання коментарів та додавання від свого імені медіа файлів.

**Мета роботи:** закріплення знань по роботі з об’єктно-орієнтованим програмуванням на мові програмування Java і засвоювання Java під операційну систему Android. А також по роботі з компонентами які призначені для баз даних та роботі з графічним інтерфейсом користувача під операційну систему Android.

**Об’єкт дослідження:** компоненти та методи які призначені для роботи з базою даних та backend, опрацювання даних у таблицях баз даних, а також організація роботи застосунку в операційній системі Android.

**Предмет дослідження:** методи роботи з базою даних та backend за допомогою мови програмування Java в середовищі Android Studio на платформі Android. Та основні функції, методи та запити мови для роботи з базами даних SQL і технологією Parse.

**Аналіз завдання та вибір засобів розробки програмного продукту**

* 1. **Опис предметної області та постановка завдання**

Об’єктно – орієнтоване програмування (скорочено ООП) і породжене ним об’єктно – орієнтоване проектування – це досконалий новий підхід до побудови складних (і не дуже складних) програм і систем. Цей підхід зародився в таких мовах програмування, як Ада, Smalltalk, C++, Borland Pascal, Java, C#. До появи ООП панувало процедурне програмування. Тоді основою програм були функції та процедури, тобто дії. Розробник визначав, які дії, які функції потрібні йому для виpiшення поставленої задачі, і peaлізовував ці функції у єдину програму. Програма звичайно мала досить чіткий алгоритм роботи – послідовність операцій.

В об’єктно-орієнтованому програмуванні і проектуванні головним не є процедура, не дія, об'єкт. Узагалі говорячи, такий підхід представляється досить природнім, оскільки в peaльному світі ми маємо справу саме з об'єктами (людьми, предметами, технічними пристроями), взаємодіючими один з одним. Взаємодія користувача з комп’ютерною програмою – це теж взаємодія двох об'єктів – програми і людини, що обмінюються один з одним визначеними повідомленнями.

Прикладна програма (додатки), побудована по принципах об'єктної орієнтації – це не послідовність якихось операторів, які слідують по певному алгоритму. Об’єктно-орієнтована програма – це сукупність об'єктів і способів їхньої взаємодії. Окремим (і головним) об'єктом при такому підході в багатьох випадках можна вважати користувача програми. Обмін між об'єктами відбувається за допомогою повідомлень.

Parse був мобільним бекендом як сервісною платформою, спочатку розробленою постачальником Parse Inc.

Parse Server працює з платформою веб-додатків Express. Його можна додати до існуючих веб-програм або запустити самостійно.

Parse використовується для push-сповіщень, взаємодію об’єктів будь-то медіа чи іншого виду. Завдяки цій технології можна створювати backend для додатку Android, що дозволяє організувати інтернет взаємодію.

**1.2. Засоби розробки програмного продукту**

**Android Studio —** інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс (англ. Ellie Powers). 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0

**Android Studio** прийшло на зміну плаґіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), macOS і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Wear OS, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена ​​колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

Parse platfrom використовується для швидшого створювання програм за допомогою зберігання об’єктів і файлів, автентифікація користувача, push-повідомлення, інформаційна панель і багато іншого.

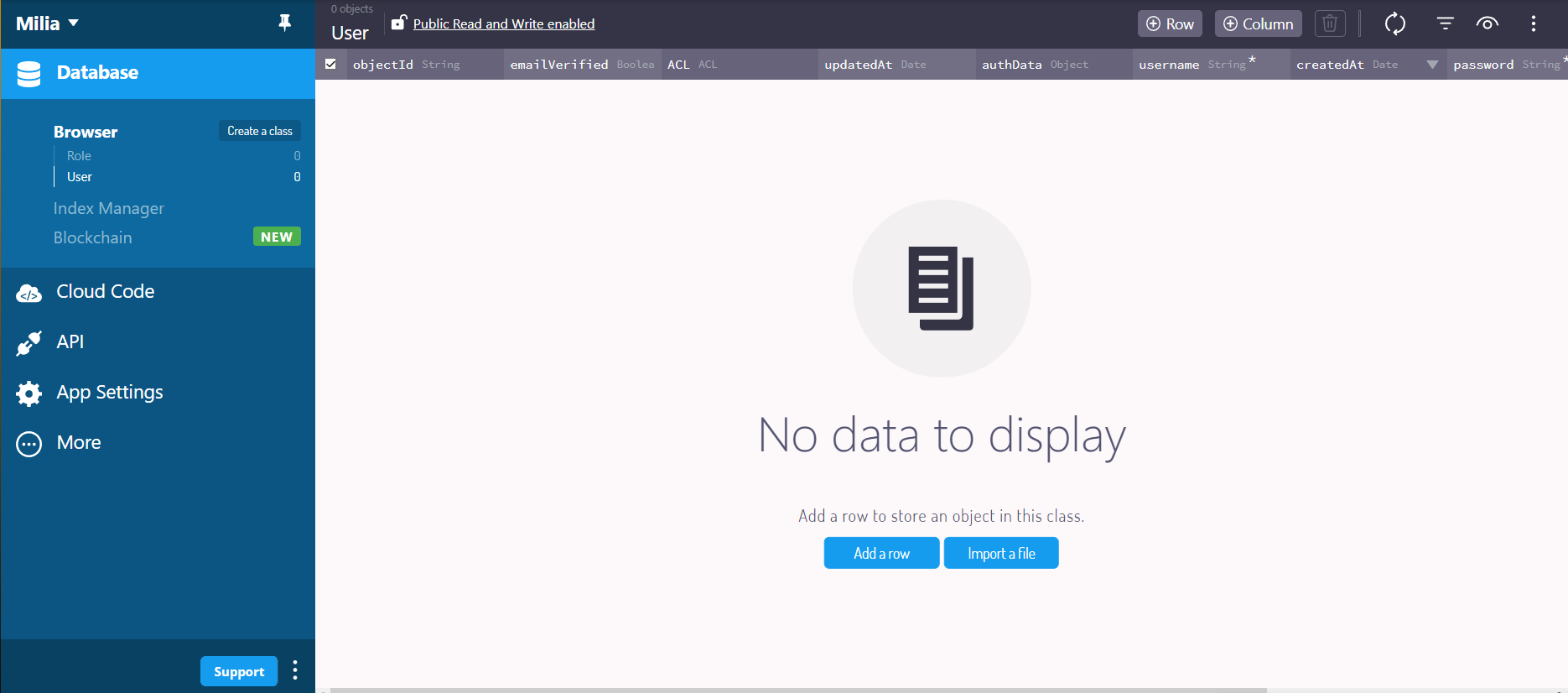


Рис.1.1 Клас “Панель керування об’єктами та іншими елементами Parse Platform”

**2. ПРОЕКТУВАННЯ ДІАГРАМ ТА АЛГОРИТМІВ ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ**

**2.1 Проектування діаграм потоків даних**

Всі дані будуть зберігатися в кількох класах Parse. На платформі Parse реалізовані такі класи:

* User
* Sessions
* Followers
* Image

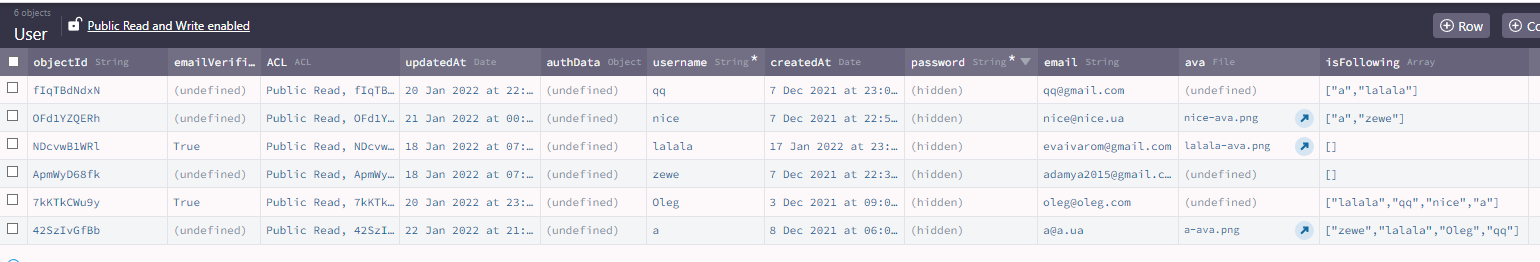


Рис.2.1 Клас “User”

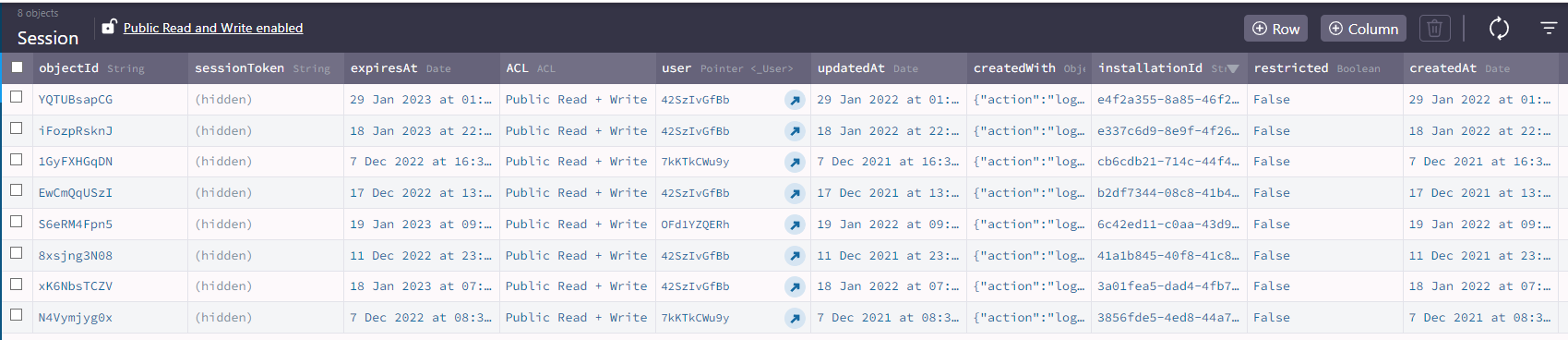
****

Рис.2.2 Клас “Sessions”



Рис.2.3 Клас “Followers”

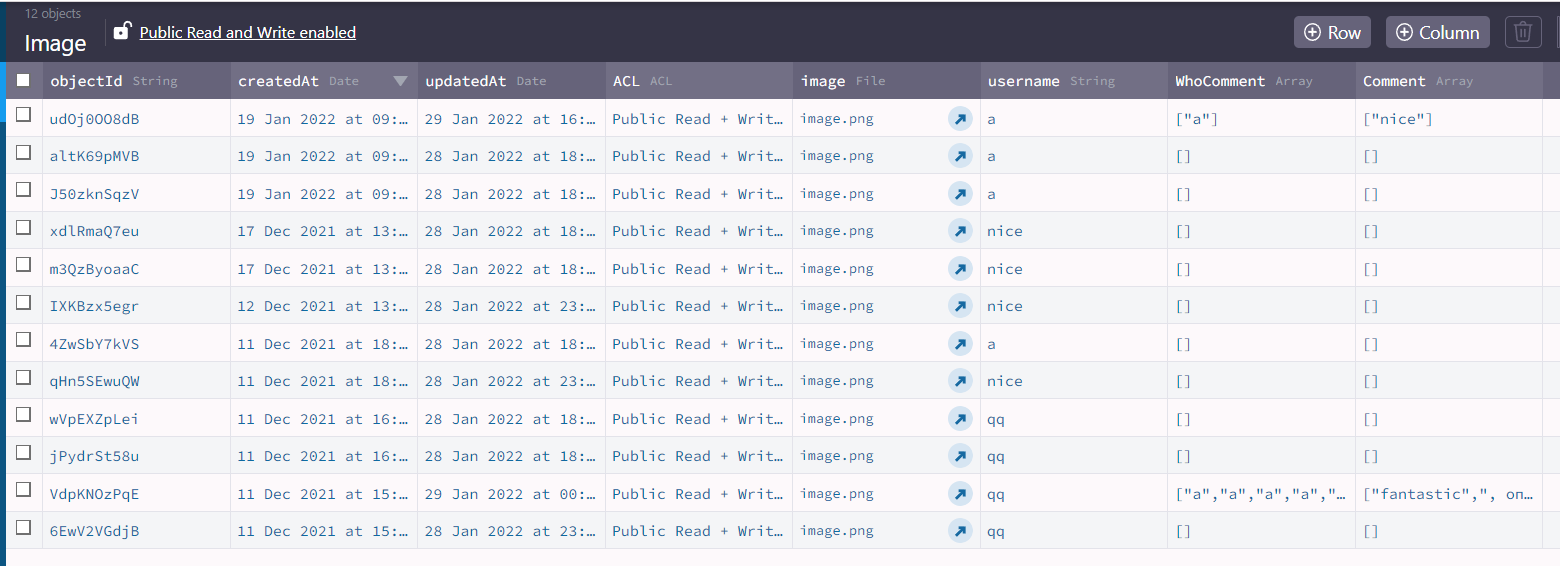


Рис.2.4 Клас “Image”

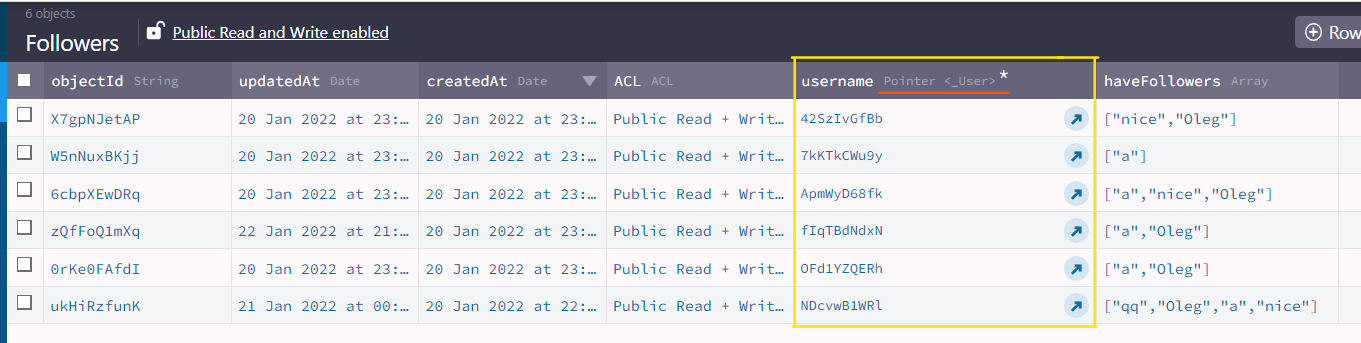
Зв’язки між таблицями створюються за допомогою полів з типом Pointer. В данному випадку, як на рисунку 2.5 у поле username поміщається об’єкт з класу User, для створення зв’язку між данною таблицею Followers та таблицею User.

Рис. 2.5 Зв’язки між таблицями

**2.2 Розробка алгоритму вирішення завдання**

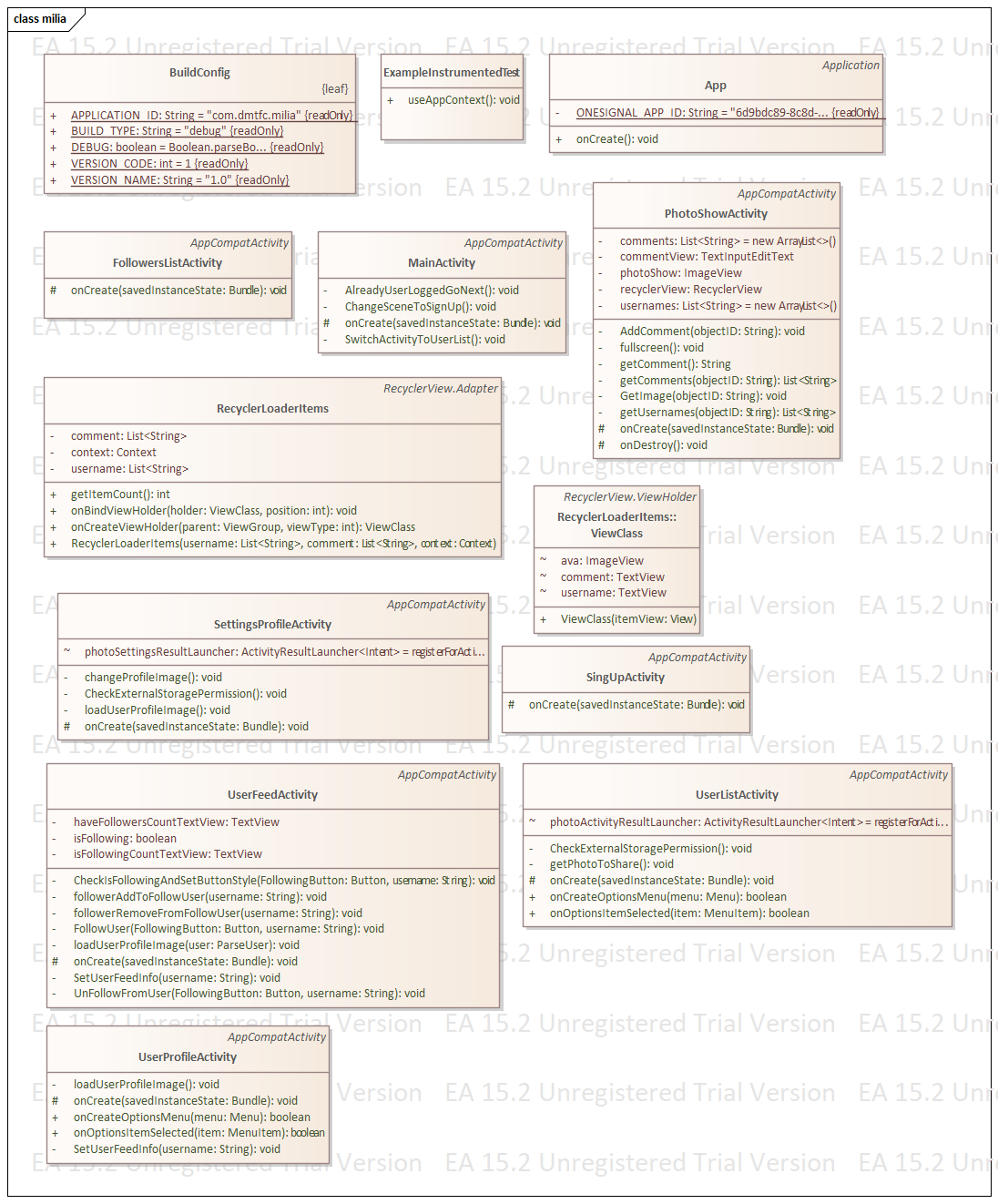
Рис 2.6 – UML-діаграма проекту.

 Рис 2.6 – UML-діаграма зв'язків данних.

**3. РОЗРОБКА СТРУКТУРИ ТА МОДУЛІВ**

**3.1. СТРУКТУРА ПРОГРАМНОГО ПРОДУКТУ.**

Програма складається з кількох активностей та класів, які забезпечують коректну роботу програми.

* **App (app.java)** – Клас певних специфічних налаштувань для OneSignal та Parse
* **FollowersListActivity (FollowersListActivity.java)** – Клас, який керує activity\_followers list (activity\_followers\_list.xml), відображає підписки або читачів певного користувача.
* **MainActivity (MainActivity.java)** – Головний клас, де відбувається старт програми та на main\_activity (main\_activity.xml) відбувається авторизація.
* **PhotoShowActivity (PhotoShowActivity.java)** – Клас, який керує activity\_photo\_show (activity\_photo\_show.xml), відображає фото на повний екран та написання коментарів під конкретне фото.
* **RecyclerLoaderItems (RecyclerLoaderItems.java)** – Клас, який керує RecyclerView на activity\_photo\_show, відповідає за відображення коментарів (коментарі виглядають, як описано в comments.xml).
* **SettingsProfileActivity (SettingsProfileActivity.java)** – Клас, який керує activity\_settings\_profile (activity\_settings\_profile.xml), де відображаються певні налаштування та можна змінити аватар користувача.
* **SingUpActivity (SingUpActivity.java)** – Клас, який керує activity\_sing\_up (activity\_sing\_up.xml), де відбувається реєстрація нового користувача.
* **UserFeedActivity (UserFeedActivity.java)** – Клас, який керує activity\_user\_feed (activity\_user\_feed.xml), опрацьовує і відображає фотографії користувача і надає можливість стежити чи відписатись від користувача, а також переглянути кількість постів, читачів, підписок і хто входить до читачів та підписок.
* **UserListActivity (UserListActivity.java)** – Клас, який керує activity\_user\_list (activity\_user\_list.xml), відображає список усіх користувачів в системі.
* **UserProfileActivity (UserProfileActivity.java)** - Клас, який керує activity\_user\_profile (activity\_user\_profile.xml), відображає профіль поточного користувача.
* **LookingForUsersActivity** **(LookingForUsersActivity.java)** – Клас, який керує activity\_looking\_for\_users (activity\_looking\_for\_users.xml), дозволяє робити пошук конкретного користувача по базі.

**3.2. Функції частини програмного забезпечення.**

Програма виконує наступні функції:

1. Процес авторизації
2. Процес реєстрації
3. Процес завантаження фото
4. Процес опрацювання коментарів
5. Процес додавання коментарів
6. Процес авторизації:

Button AuthButton = (Button) findViewById(R.id.*authButton*);  
  
final EditText editTextLogin = (EditText) findViewById(R.id.*editTextLoginRegister*);  
final EditText editTextPassword = (EditText) findViewById(R.id.*editTextPassword*);  
  
*// Check if logged*if (ParseUser.*getCurrentUser*() != null) {  
 *// check if authenticated* if (ParseUser.*getCurrentUser*().isAuthenticated()) {  
 *// If yes go on List of users layout (UserListActivity)* AlreadyUserLoggedGoNext();  
 *// Set external unique if for push messages when logged* String userID = ParseUser.*getCurrentUser*().getObjectId();  
 OneSignal.*setExternalUserId*(userID);  
 }  
}  
  
*// check if login or password field is empty*if (editTextLogin.getText().toString() == "" || editTextPassword.getText().toString() == "") {  
Toast.*makeText*(getApplicationContext(), "Поле логін або пароль - пусте.", Toast.*LENGTH\_SHORT*).show();  
Toast.*makeText*(getApplicationContext(), "Введіть логін і пароль, будь ласка", Toast.*LENGTH\_SHORT*).show();  
} else {  
*//Auth button*AuthButton.setOnClickListener(new View.OnClickListener() {  
@Override  
public void onClick(View view) {  
*//Auth in parse system*ParseUser.*logInInBackground*(  
editTextLogin.getText().toString(),  
editTextPassword.getText().toString(),  
new LogInCallback() {  
@Override  
public void done(ParseUser parseUser, ParseException e) {  
if (parseUser != null) {  
Toast.*makeText*(getApplicationContext(), "Авторизація успішна!", Toast.*LENGTH\_SHORT*).show();  
Log.*i*("Auth", "Sign In: OK!");  
*// Set external unique if for push messages when logging*String userID = ParseUser.*getCurrentUser*().getObjectId();  
OneSignal.*setExternalUserId*(userID);  
  
*// Go to User List Activity*SwitchActivityToUserList();  
} else {  
Toast.*makeText*(getApplicationContext(), e.getLocalizedMessage(), Toast.*LENGTH\_SHORT*).show();  
e.printStackTrace();  
}}});}});}

1. Процес реєстрації:

final EditText emailText = (EditText) findViewById(R.id.*editTextEmailAddressRegister*);  
final EditText loginText = (EditText) findViewById(R.id.*editTextLoginRegister*);  
final EditText passwordText = (EditText) findViewById(R.id.*editTextPasswordRegister*);  
  
Button registerButton = (Button) findViewById(R.id.*registerButton*);  
  
*//Check if edits are empty*if (emailText.getText().toString() == "" || loginText.getText().toString() == "" || passwordText.getText().toString() == "") {  
Toast.*makeText*(getApplicationContext(), "Введіть в усі поля дані", Toast.*LENGTH\_SHORT*).show();  
} else {  
*//Process of registration*registerButton.setOnClickListener(new View.OnClickListener() {  
@Override  
public void onClick(View view) {  
ParseUser.*logOut*(); *// Log Out. because maybe user is logged*ParseUser user = new ParseUser(); *//Create new user  
//Add info about user*user.setEmail(emailText.getText().toString());  
user.setUsername(loginText.getText().toString());  
user.setPassword(passwordText.getText().toString());  
  
*//Registration on server*user.signUpInBackground(new SignUpCallback() {  
@Override  
public void done(ParseException e) {  
if (e == null) {  
Toast.*makeText*(getApplicationContext(), "Ви успішно зареєструвались " + loginText.getText().toString(), Toast.*LENGTH\_SHORT*).show();  
Log.*i*("SingUp", "Success registration of " + loginText.getText().toString());  
ParseUser.*logOut*();  
startActivity(new Intent(SingUpActivity.this, MainActivity.class));  
finish();  
Toast.*makeText*(getApplicationContext(), "Авторизуйтесь, будь ласка", Toast.*LENGTH\_SHORT*).show();  
} else {  
*//Error of registration*Log.*e*("SingUp", "Error in Sing Up!");  
Toast.*makeText*(getApplicationContext(), e.getMessage(), Toast.*LENGTH\_SHORT*).show();  
 e.printStackTrace();  
 } } }); } });}

1. Процес завантаження фотографій:

*/\*\*  
 \* Check access to External Storage.  
 \* Than Start process of sharing photo ( Method: getPhotoToShare(); )  
 \*/*private void CheckExternalStoragePermission() {  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*M*) {  
 if (checkSelfPermission(Manifest.permission.*READ\_EXTERNAL\_STORAGE*) != PackageManager.*PERMISSION\_GRANTED*) {  
 requestPermissions(new String[]{Manifest.permission.*READ\_EXTERNAL\_STORAGE*}, 1);  
 } else {  
 getPhotoToShare();  
 }}}

*/\*\*  
 \* Start activity of getting image and than  
 \* send it on server.  
 \*/*private void getPhotoToShare() {  
 Intent intent = new Intent(Intent.*ACTION\_PICK*, MediaStore.Images.Media.*EXTERNAL\_CONTENT\_URI*);  
 photoActivityResultLauncher.launch(intent);  
}

*// You can do the assignment inside onAttach or onCreate, i.e, before the activity is displayed*ActivityResultLauncher<Intent> photoActivityResultLauncher = registerForActivityResult(  
 new ActivityResultContracts.StartActivityForResult(),  
 new ActivityResultCallback<ActivityResult>() {  
 @Override  
 public void onActivityResult(ActivityResult result) {  
 if (result.getResultCode() == Activity.*RESULT\_OK*) {  
 *// There are no request codes* Intent data = result.getData();  
  
 *//do some operation* if (data != null) {  
  
 Uri selectedImage = data.getData();  
 try {  
 Log.*i*("Image", "Image Selected!");  
  
Bitmap bitmap = MediaStore.Images.Media.*getBitmap*(getContentResolver(), selectedImage);  
ByteArrayOutputStream stream = new ByteArrayOutputStream();  
bitmap.compress(Bitmap.CompressFormat.*PNG*, Integer.*parseInt*(getString(R.string.*png\_quality\_upload*)), stream);  
byte[] byteArray = stream.toByteArray();  
ParseFile file = new ParseFile("image.png", byteArray);  
ParseObject object = new ParseObject("Image");  
object.put("image", file);  
object.put("username", ParseUser.*getCurrentUser*().getUsername());  
object.saveInBackground(new SaveCallback() {  
@Override  
public void done(ParseException e) {  
if (e == null) {  
Toast.*makeText*(UserListActivity.this, "Зображення було додане!", Toast.*LENGTH\_SHORT*).show();  
Log.*i*("Add Photo", "Success photo added");  
} else {  
Toast.*makeText*(UserListActivity.this, "Помилка: " + e.getLocalizedMessage(), Toast.*LENGTH\_SHORT*).show();  
Log.*e*("Add Photo", "Error with add photo");  
e.printStackTrace();  
}}});} catch (Exception e) {  
Log.*e*("Image", "ERROR:" + e.getMessage());  
e.printStackTrace();}}}}});

1. Процес опрацювання коментарів:

public class RecyclerLoaderItems extends RecyclerView.Adapter<RecyclerLoaderItems.ViewClass> {  
 private List<String> username;  
 private List<String> comment;  
 private Context context;  
  
 public RecyclerLoaderItems(List<String> username, List<String> comment, Context context) {  
 this.username = username;  
 this.comment = comment;  
 this.context = context;  
 }  
  
 @NonNull  
 @Override  
 public ViewClass onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
View view = LayoutInflater.*from*(parent.getContext()).inflate(R.layout.*comments*, parent, false);  
 return new ViewClass(view);  
 }

*/\*\*  
 \* Setting info like username, comments, images in right place  
 \* @param holder  
 \* @param position  
 \*/* @Override  
 public void onBindViewHolder(@NonNull ViewClass holder, int position) {  
 */\* Sets username and comment \*/* holder.username.setText(username.get(position));  
 holder.comment.setText(comment.get(position));  
  
 */\* Sets ava for them \*/* ParseQuery<ParseUser> imageQuery = ParseUser.*getQuery*();  
 imageQuery.whereEqualTo("username", username.get(position));  
 try {  
 ParseObject user = imageQuery.getFirst();  
 if (user != null) {  
 ParseFile file = (ParseFile) user.get("ava");  
 if (file != null) {  
 file.getDataInBackground(new GetDataCallback() {  
 @Override  
 public void done(byte[] data, ParseException e) {  
 if (e == null && data != null) {  
Bitmap bitmap = BitmapFactory.*decodeByteArray*(data, 0, data.length);  
 holder.ava.setImageBitmap(bitmap);  
 }  
 }  
 });  
 }  
 }  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public int getItemCount() {  
 return username.size();  
 }  
  
  
 public class ViewClass extends RecyclerView.ViewHolder {  
 TextView username;  
 TextView comment;  
 ImageView ava;  
  
 public ViewClass(@NonNull View itemView) {  
 super(itemView);  
 username = itemView.findViewById(R.id.*usernameCommentTextView*);  
 comment = itemView.findViewById(R.id.*commentTextView*);  
 ava = itemView.findViewById(R.id.*AvaImageComment*);  
 }}}

1. Процес додавання коментарів

commentView = findViewById(R.id.*CommentEditText*);

Button sendComment = findViewById(R.id.*SendCommentButton*);  
sendComment.setOnClickListener(view -> AddComment(objectID));

*/\*\*  
 \* Send user's comment to backend.  
 \* @param objectID Object ID of open image  
 \*/*private void AddComment(String objectID) {  
ParseQuery<ParseObject> commentsQuery = ParseQuery.*getQuery*("Image");  
commentsQuery.getInBackground(objectID, new GetCallback<ParseObject>() {  
@Override  
public void done(ParseObject object, ParseException e) {  
if (e == null) {  
object.add("WhoComment", ParseUser.*getCurrentUser*().getUsername());  
String userComment = getComment();  
object.add("Comment", userComment);  
object.saveInBackground(new SaveCallback() {  
@Override  
public void done(ParseException e) {  
if (e == null) {  
Toast.*makeText*(PhotoShowActivity.this, "Успішно доданий коментар", Toast.*LENGTH\_SHORT*).show();  
commentView.setText("");  
} else {  
Toast.*makeText*(PhotoShowActivity.this, "Не успішно доданий коментар", Toast.*LENGTH\_SHORT*).show();  
Log.*e*("Error add comment", "Unsuccessful add comment: " + e.getMessage());  
e.printStackTrace();}}});}}});}

Щоб підключитись до Parse необхідно:

1. Запустити сервер Parse або використати ресурс *back4app.com,* який допоможе організувати сервер.

2. Потрібно підключити модуль parse:

В файлі build.gradle (Module:app) потрібно налаштувати пункт repositories{…}.



Рис.3.1 Підключення репозиторіїв до проекту в Android Studio

CompileSdkVersion має бути 27 або вище, щоб бути сувмістима з Parse SDK.

3. Далі в build.gradle (Module:app) в пункті dependencies необхідно підключити саму Parse SDK.



Рис.3.2 Підключення Parse SDK до проекту в файлі build.gradle (Module:app)

**GridLayout** використовують для відображення даних у вигляді сітки або таблиці, вона має такі властивості як колонки та рядки.

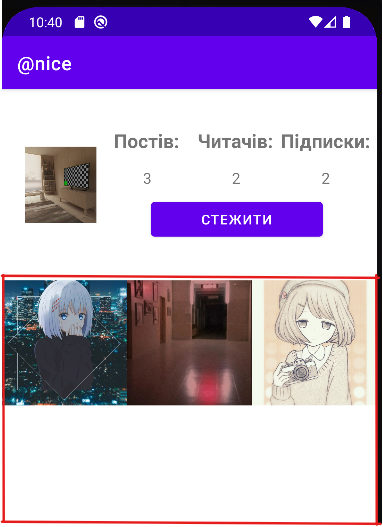


Рис.3.3 Компонент **GridLayout** в червоній рамці

За допомогою **Parse SDK** та **Android** можна виводити різні дані в **GridLayout**. На прикладі рисунку 3.3 видно, що можна додавати об’єкти такі як **ImageView**.



Рис.3.4 Код додавання **ImageView** до **GridLayout**

**4. Керівництво користувача**

Після встановлення та запуску програми на екрані у користувача появиться “Активність Авторизації”. На формі представлені 2 поля для вводу логіна та паролю, кнопка авторизації та реєстрації.

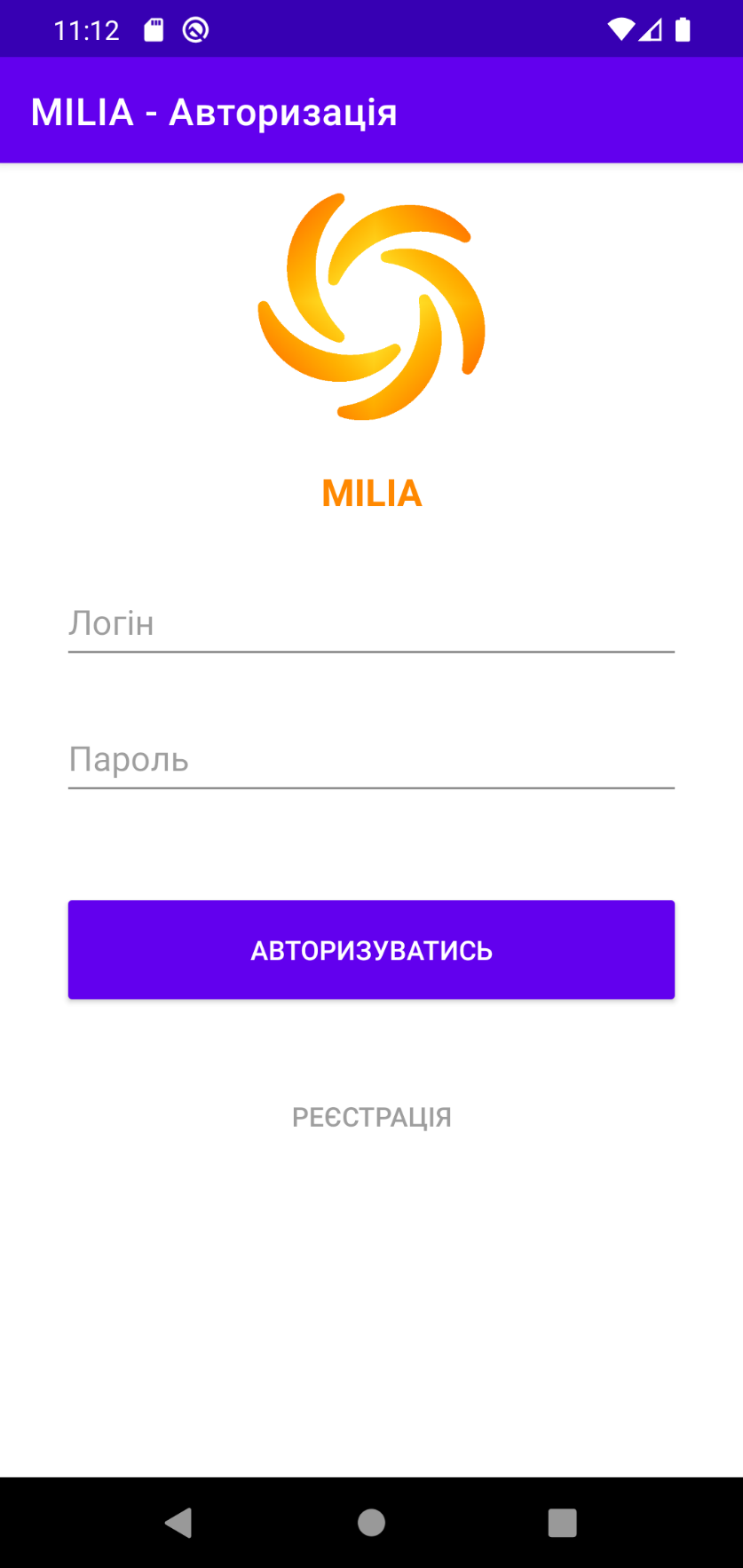


Рис.4.1 Активність авторизації в додатку.

• Демонстрація актовності реєстрації:

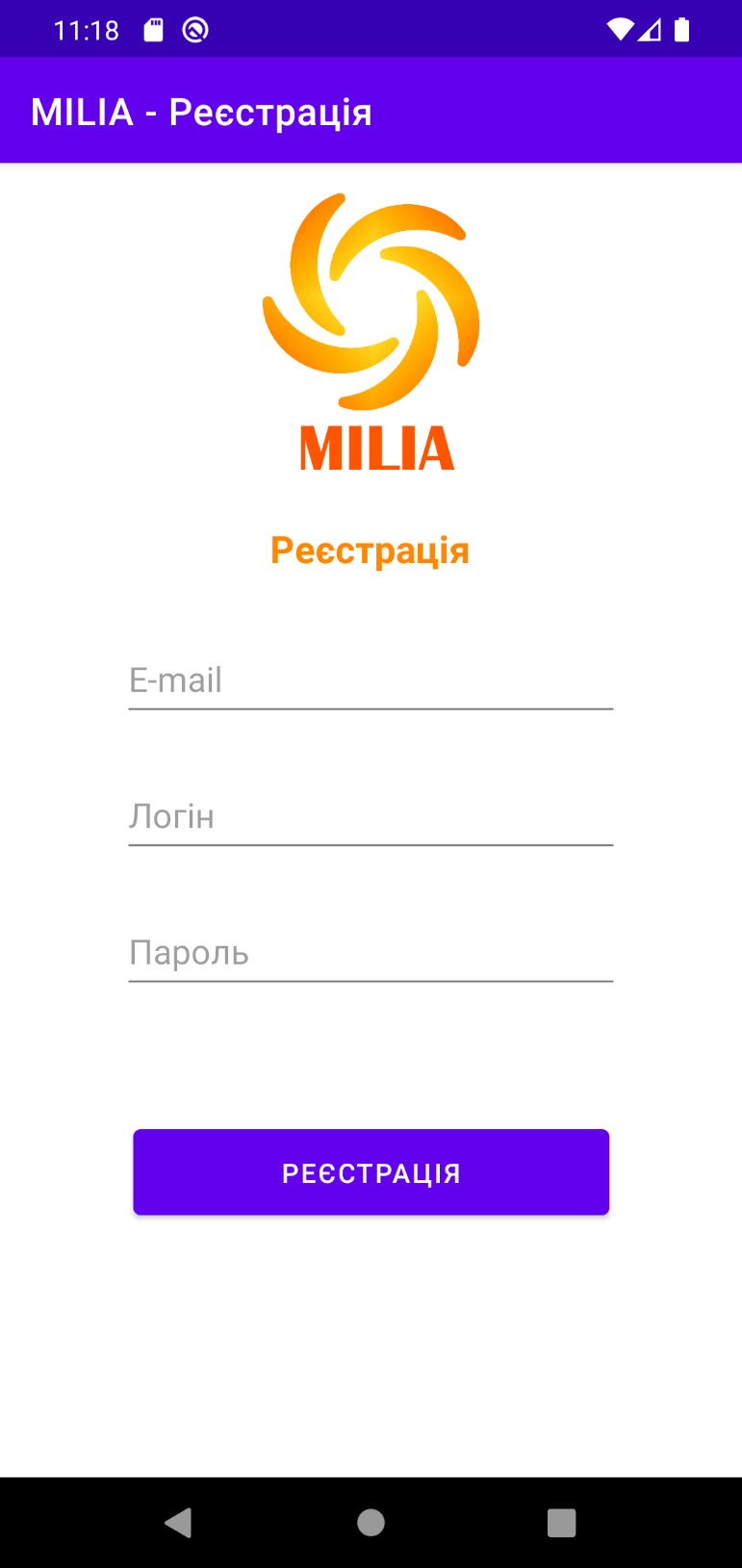
****

Рис.4.2 Активність реєстрації

Якщо ви вперше в програмі, то варто пройти реєстрацію, так як в базі даних немає вас. В першому полі **«E-mail»** потрібно ввести вашу електрону пошту. В наступних полях потрібно ввести **«Логін»** та **«Пароль»**, для майбутньої авторизації. Тепер потрібно нажати кнопку «**Реєстрація**». З’явиться відповідне повідомлення якщо все успішно.

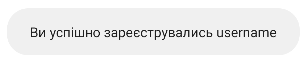


Рис.4.3 Повідомлення про успішну реєстрацію.

**•** Демонстрація активності зі списком користувачів:

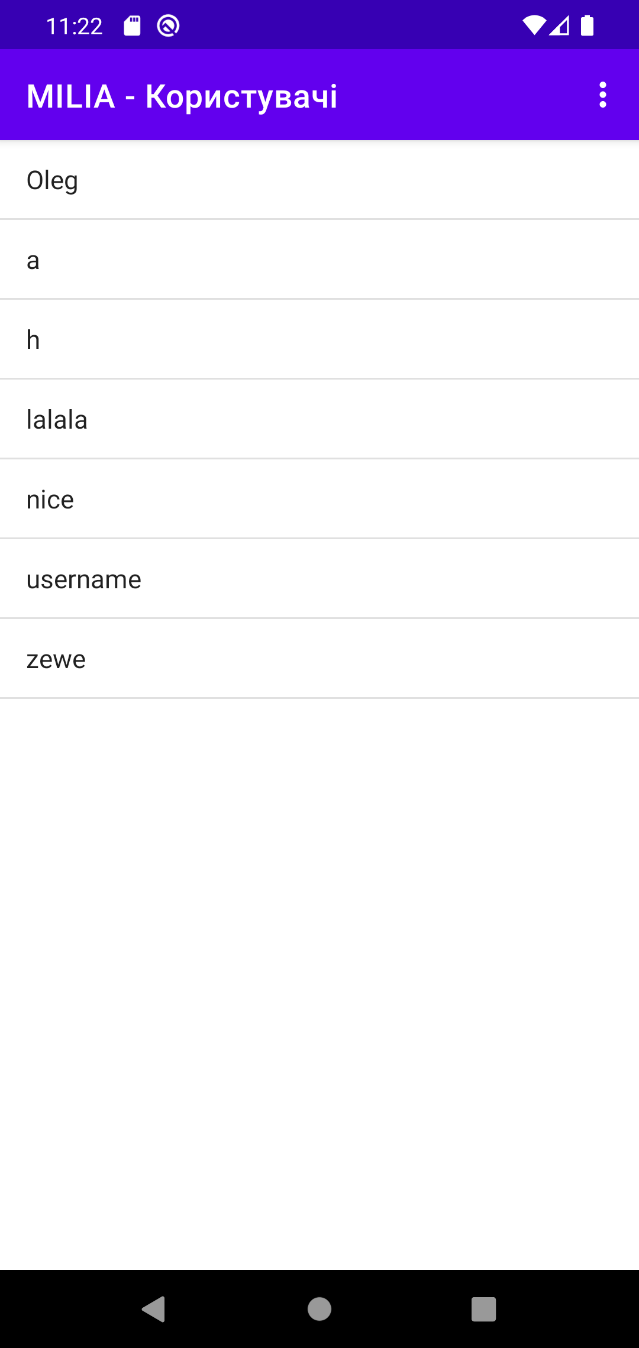
****

Рис.4.4 Активність списку користувачів.

При натисканні на якогось з **користувачів** відкриється активний профіль данного користувача, де можна буде переглянути пости користувача за ким він стежить та хто стежить за ним, а також можливість стежити за користувачем.

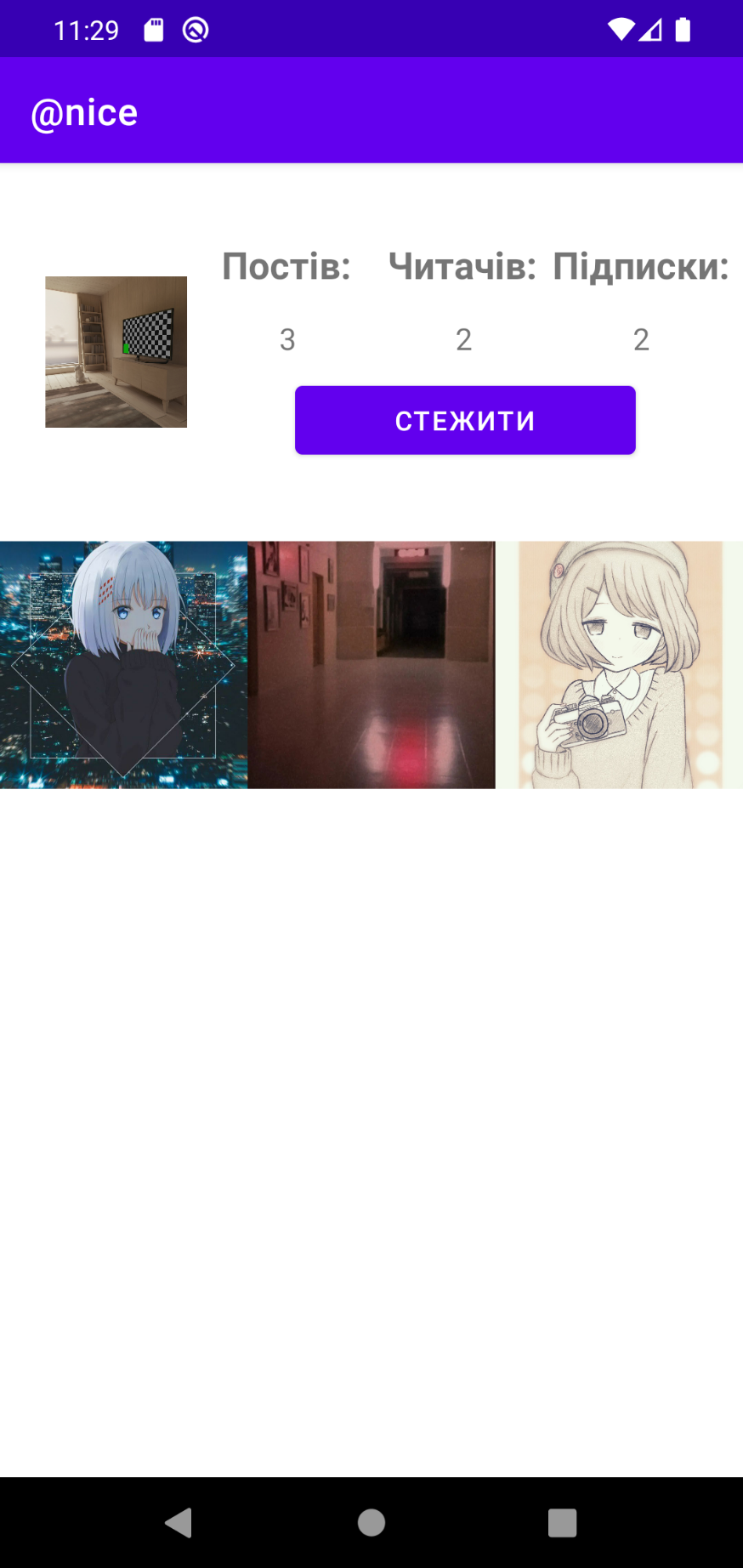


Рис.4.5 Активність профіля обраного користувача.

У профілі користувача ми можемо побачити, його викладені фото, аватарку, деяку інформацію та можливість стежити та навпаки. Якщо ви бажаєте стежити за користувачем, то можете нажати на кнопку «**Стежити**» і тоді вона зміниться на сірий колір і буде написано «**Відстежується**» з чого буде зрозуміло, що ви стежите за користувачем і при повторному нажимі на кнопці ви перестанете стежити за користувачем.

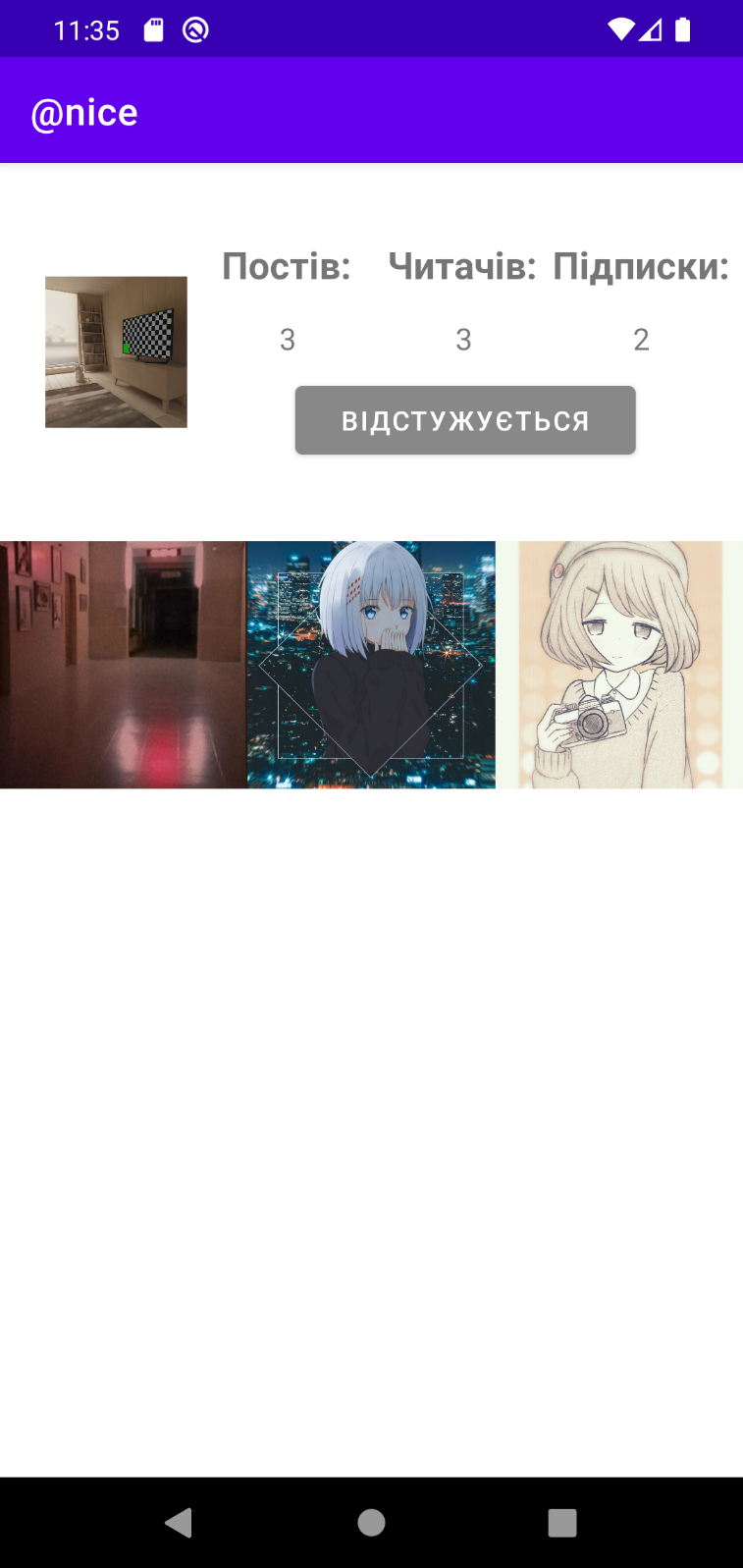


Рис.4.6 Вигляд кнопки коли ви стежите за користувачем.

При нажимі на цифру з «**Читачами**» чи «**Підписки**» то у вас є можливість переглянути читачів данного користувача.

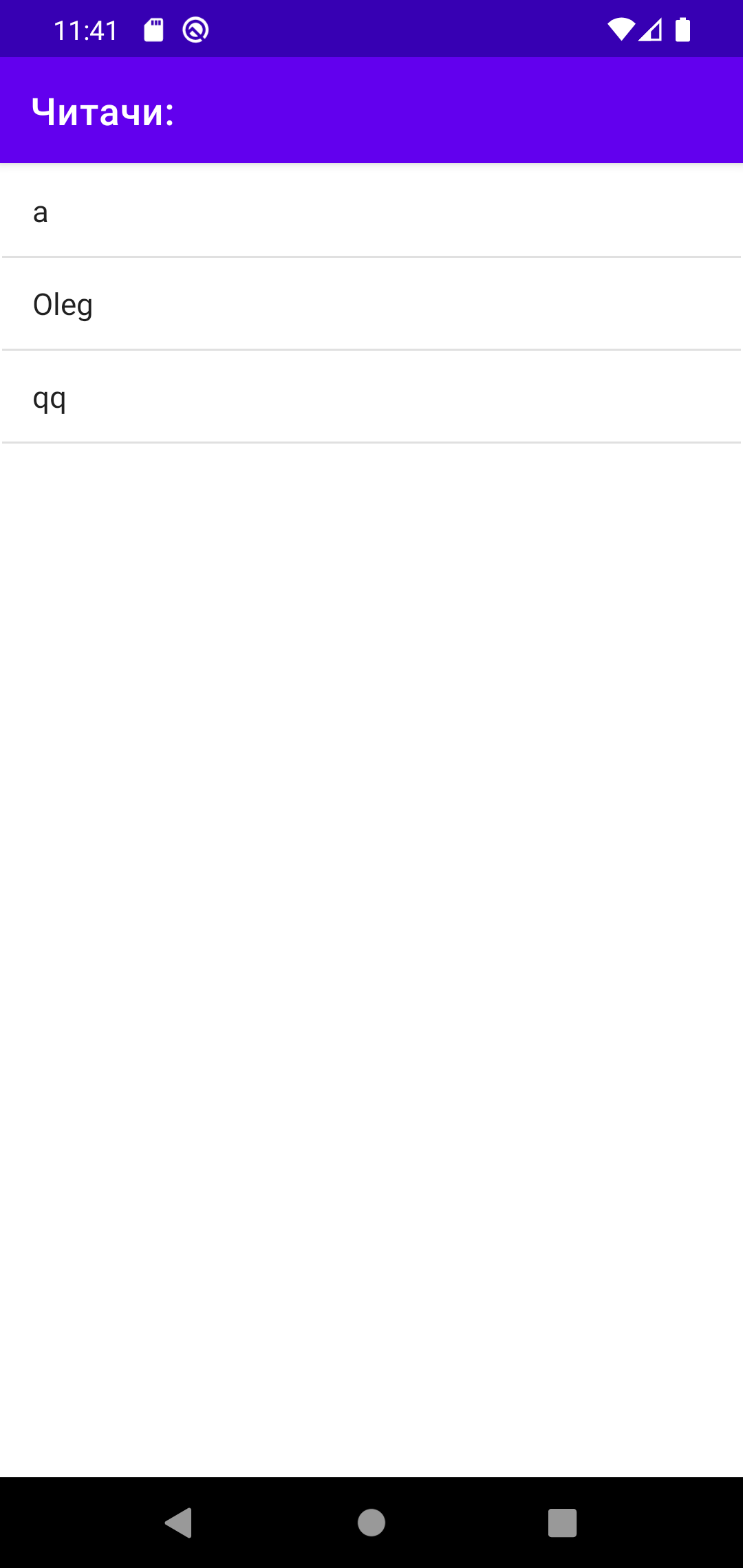


Рис.4.7 Вигляд активності з читачами.



Рис.4.8 Вигляд активності з підписками.

Повернувшись на форму з користувачем у вас є можливість переглянути його фото у більшому розмірі для цього потрібно лиш обрати на потрібне фото. Відкриється нова активність з фотографією та нижче буде розміщуватись секція з коментарями.

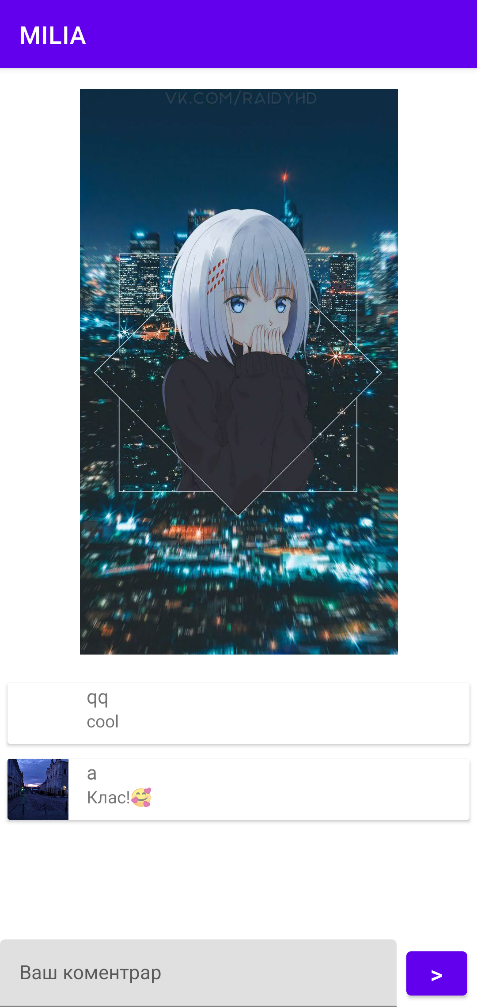
****

Рис.4.8 Вигляд активності з відкритим фото та коментарями.

Нижче в полі «**Ваш коментар**» можна ввести коментар і нажмаючи на кнопку  надіслати коментар. Про успішне додання коментаря з’явиться додаткове повідомлення.



Рис.4.9 Повідомлення про успішно доданий коментар.

Для закриття фотографії потрібно ще раз на ній нажати, тоді вона закриється і вернеться до форми з профілем.

Тепер давайте повернемось до активності з користувачами і у правому верхньому куті є меню. ()

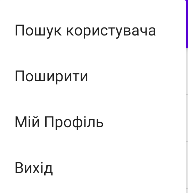


Рис.4.9 Меню на активності з користувачами.

* Поширити – за допомогою цього пункту ви можете додати свою фотографію. Після натискання при першому запуску запросить дозвіл на отримання доступу до сховища.

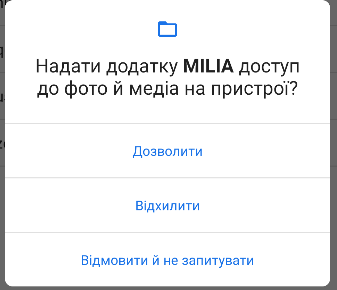


Рис.4.10 Меню на активності з користувачами.

Далі відкриється вікно з вибором фото-файлу для відправки на сервер. Потрібно буде обрати фото і підтвердити вибір. Якщо все успішно, з’явиться наступне повідомлення:

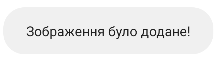


Рис.4.11 Повідомлення про успішне додання фото.

* Мій профіль – Відкриває активність з профілем поточного авторизованого користувача.

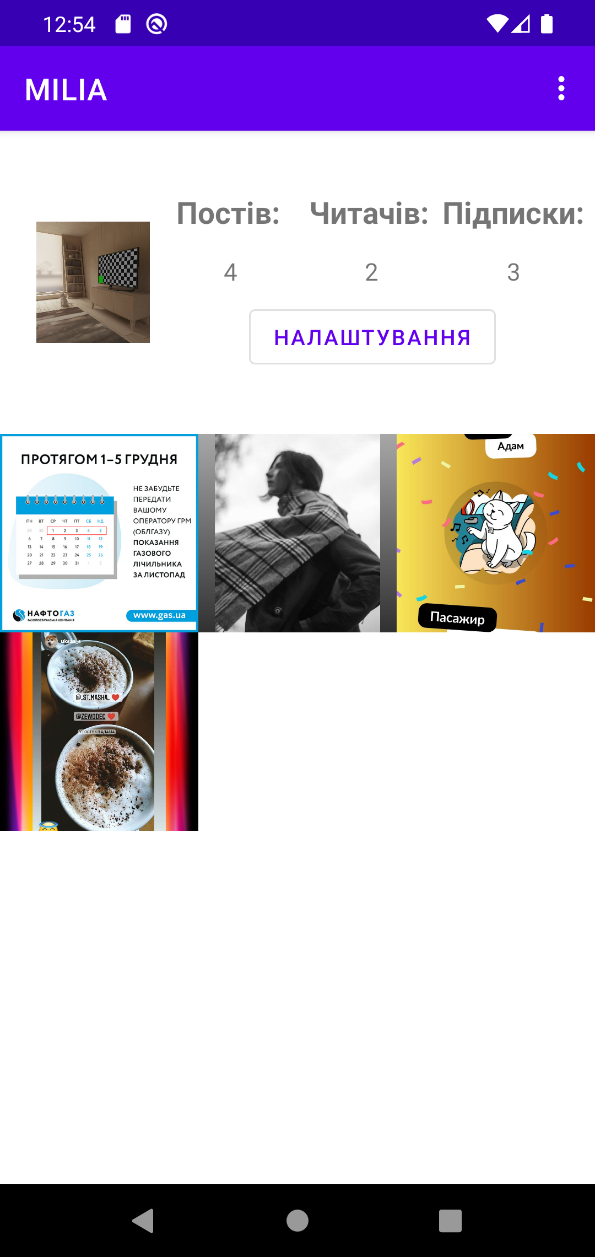


Рис.4.12 Активність «**Мій профіль**».

При обрані в пункті меню (у верхньому правмо куті) або нажимі по кнопці «**Налаштування**» ви переходити на активність «**Налаштування**».



Рис.4.12 Активність «**Налаштування**».

Тут можна змінити аватар завдяки кнопці «**Змінити світлину**». Після нажиму по ній потрібно обрати фото і з’явиться відповідне повідомлення якщо все успішно.

* Вихід – Виходить з профілю авторизовано користувача і повертає на активність «**Авторизація**»



Рис.4.13 Активність «**Пошук користувача**».

* Пошук користувача – відкриває активність, де можна ввести в поле «Ім’я шуканого користувача» і потім отримати список користувачі, які починаються відповідно до вводу користувача.

Нажавши на кнопку з іконкою «» користувач запустить процес пошуку.

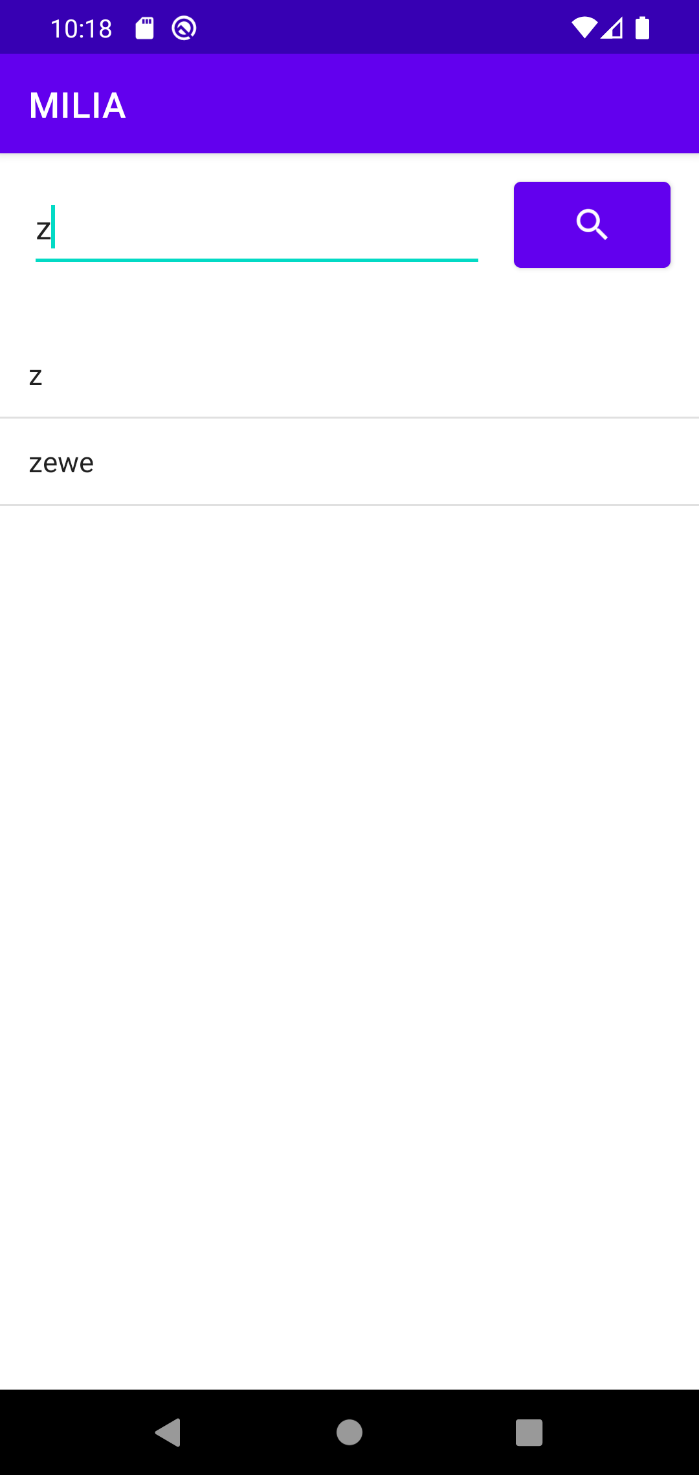


Рис.4.13 Приклад роботи пошуку користувача.

Потім на екрані в нього з’явиться користувачі, які починаються на ті символи, які увів користувач. Про успішний пошук з’явиться відповідне повідомлення.



Рис.4.14 Повідомлення про успішний пошук.

Також до програми підключена платформа **«OneSignal»**. Вона дозволяє розсилати Push-сповіщення. Для відправлення загальних Push-сповіщень необхідно авторизуватись на платформі «**OneSignal**» за адресою <https://app.onesignal.com> , де необхідно обрати власний проект в даному випадку «**MILIA**».

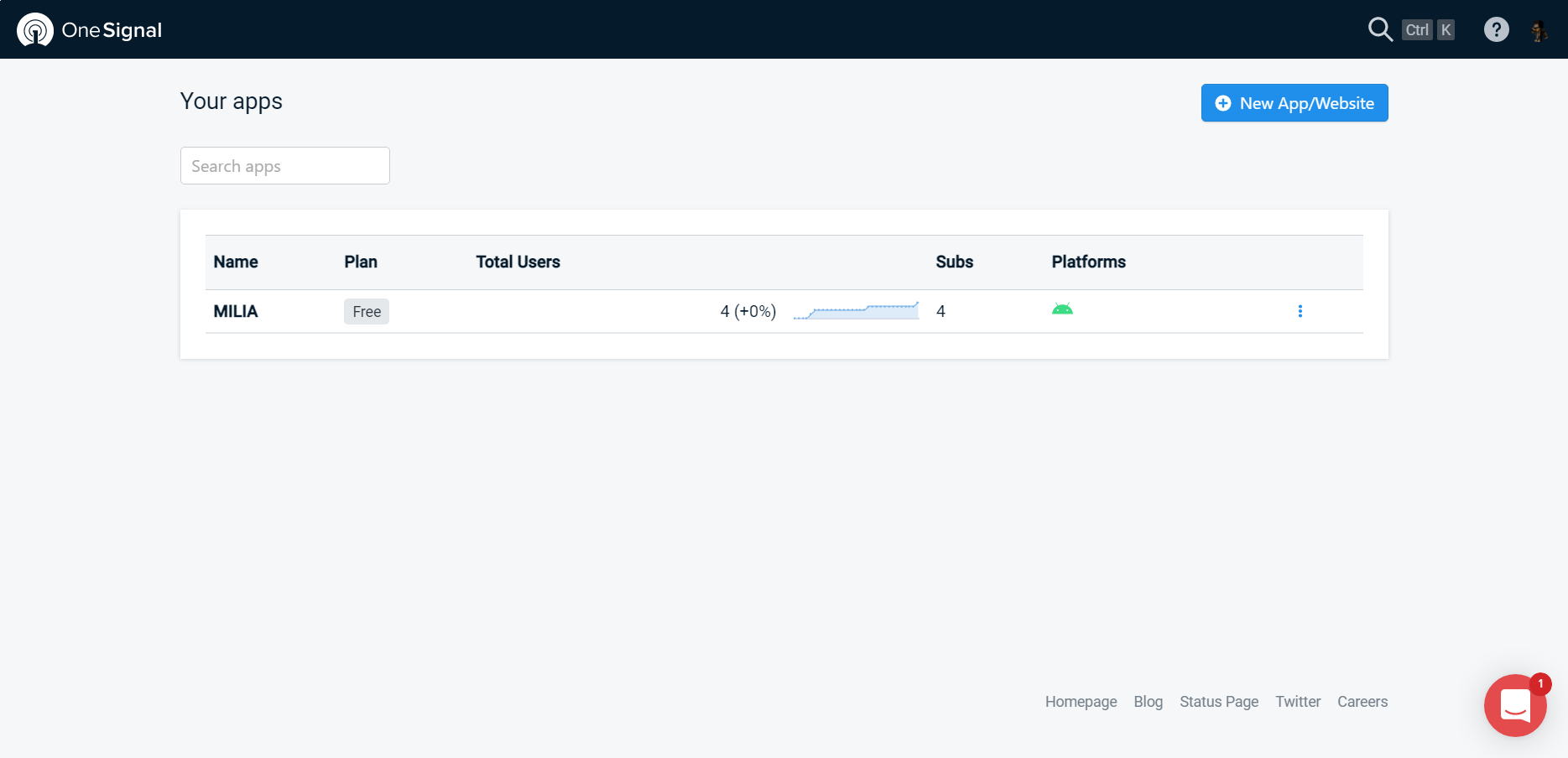


Рис.4.15 Платформа «OneSignal».

На сторінці, яка відкрилась після обрання проекту, з’явиться статистика по даній програмі, а також кнопка, яка надає можливість створювати Push-сповіщення.

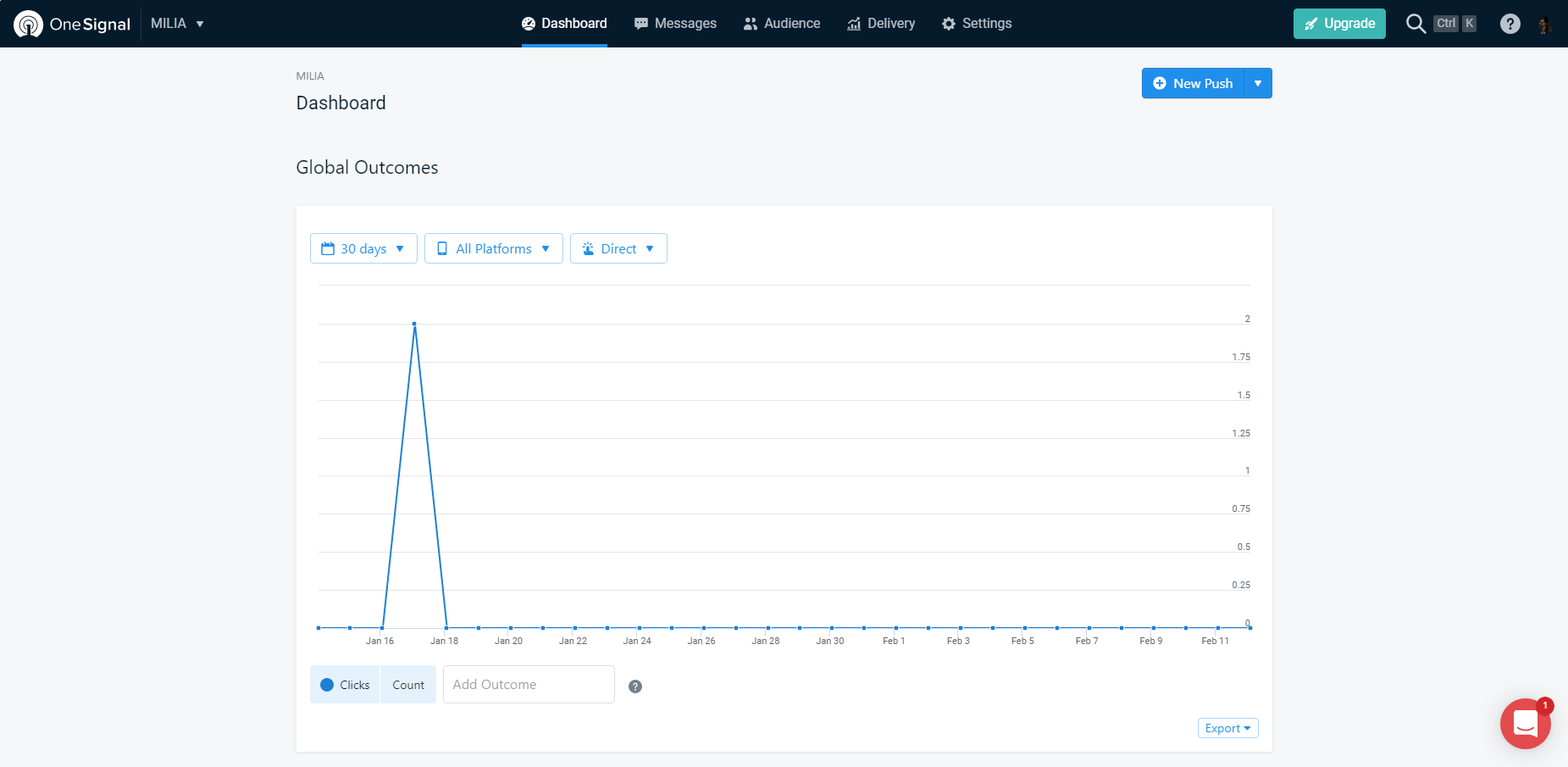


Рис.4.15 Платформа «OneSignal» з відкритим проектом «Milia».

При обранні пункту  відкриється вікно для створення нового Push-повідомлення. Де буде кілька пунктів для заповнення:

1. Message Name – де потрібно увести назву даного Push-повідомлення, особисто для себе, щоб розуміти короткий смисл про що воно буде, так як по ньому система веде статистику, таку як кількість користувачів, які відкрили чи не відкрили дане Push-повідомлення.
2. Audience – вибір аудиторії, тобто для яких осіб буде відправлене дане Push-сповіщення, наприклад, активні користувачі, не активні тощо.
3. Message – саме наповнення Push-повідомлення.
4. Delivery Schedule – пункт для обрання коли потрібно робити розсилку.
5. Message Name

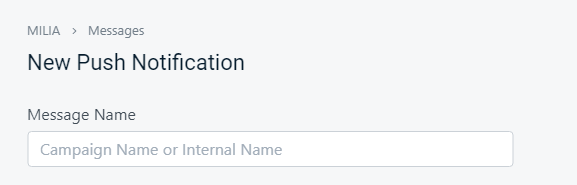


Рис.4.16 Вигляд пункту «Message Name».

В поле «Message Name» необхідно вписати робочу назву Push-сповіщення.

1. Audience

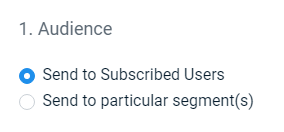


Рис.4.17 Вигляд пункту «Audience».

Обираючи пункт «Send to Subscribed Users» - Push-сповіщення відправиться до усіх, хто установив додаток, а інший варіант вибору «Send to particular segment(s)» - дозволяє настроїти, хто буде отримувати Push-сповіщення.

1. Message

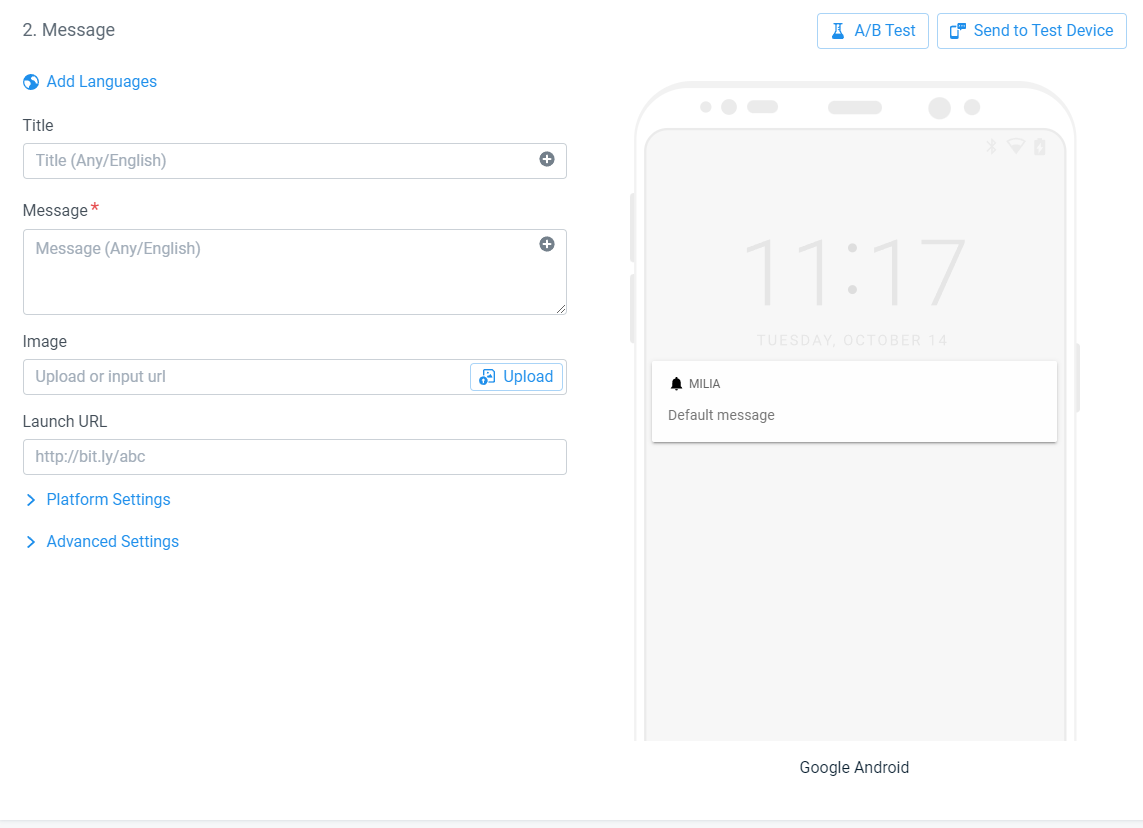


Рис.4.18 Вигляд пункту «Message».

* Title – тут вказується заголовок Push-сповіщення.
* Message – основний текст сповіщення.
* Image – надає, можливість завантажити фото, яке буде показуватись в Push-сповіщенні.
* Launch URL – задає URL адресу на яку буде переадресовано користувача, коли він натисне по Push-сповіщенню.
* Platform Settings та Advanced Settings – надають додаткові, більш глибокі налаштування.
* Add Language – надає можливість створити Push-сповіщення різними мовами, щоб відповідно до настройок користувача прийшло повідомлення на тій мові, яка обрана користувачем.
* Send To Test Device – при обранні даної функції, можна відправити дане повідомлення на пристрої для тесту.
* З права є приклад, як буде відображатись Push-повідомлення і відповідно до настройок зліва, буде динамічно змінюватись.

1. Delivery Schedule

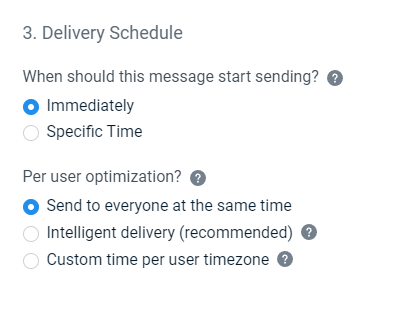


Рис.4.19 Вигляд пункту «Delivery Schedule».

* Immediately – вказує, що розпочинати потрібно від поточного моменту.
* Specific Time – вказує, коли потрібно розпочинати розсилку.
* Send to everyone at the same time – налаштування, яке вказує системі, щоб відправити всім користувача одночасно Push-повідомлення.
* Intelligent delivery (recommended) – рекомендована опція, яка відправляє Push-повідомлення в той момент, коли передбачається, що користувач буде активний.
* Custom time per user timezone – опція, яка вказує о котрій годинні повинне бути Push-повідомлення в залежності від часового поясу користувача.



Рис.4.20 Вигляд кнопок для збереження або запуску Push-повідомлення.

Кнопка «**Review and Send**» - дозволяє зробити перегляд Push-повідомлення, яке вийшло в результаті та відправити його до користувачів на телефони. Кнопка «Save as Draft» - дозволяє зберегти створене Push-повідомлення, як чернетку на потім та за потреби використати.

Також платформа «**OneSignal**» надає можливість створювати In-App повідомлення – це вікна, які з’являються у користувача при певних умовах, але їх видно на екрані не так, як Push-повідомлення. При запуску програми «Milia» з’являється вітальне повідомлення:

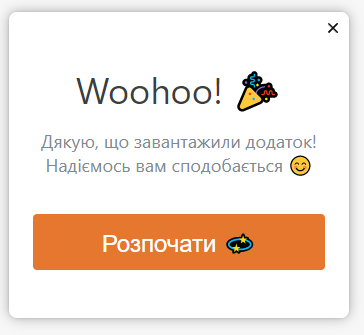


Рис.4.21 Вигляд In-App вітального повідомлення в програмі «Milia».

Відповідно є можливість по різному редагувати і використовувати In-App повідомлення та отримувати статистику, як користувачі з нею взаємодіють.

**Висновок**

В ході виконання курсової роботи було організовано програму «Milia», яка дозволяє поширювати фото та коментувати їх різним користувачам. Фото, коментарі та інформація про користувачів зберігаються в базі даних завдяки Parse. Реалізований перегляд, додавання, коментування фотографій, а також стеження за користувачами. Також реалізований процес авторизації та реєстрації. Програму було розроблено засобами мови програмування Java за допомогою середовища Android Studio. Курсова розроблялась під систему Android API 29 проте підтримка відбуваєтсья від API 21 тобто Android 5.0 Lolipop.

**Список використаної літератури**

1. Алгоритми i структура даних: Навчальний посiбник / В.М.Ткачук. – Iвано-Франкiвськ : Видавництво Прикарпатського нацiонального унiверситету iменi Василя Стефаника, 2016. 286 с
2. Мартін Роберт, Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во «Ранок» : Фабула, 2021. – 448 с.
3. Мартін Роберт, Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во «Ранок» : Фабула, 2021. – 368с.
4. Інтернет-ресурс, Back4app.com, Документація по Parse SDK - <https://www.back4app.com/docs>
5. Інтернет-ресурс, Android Developer, Документація по Android - <https://developer.android.com/docs>
6. Інтернет-ресурс, OneSignal, Документація по OneSignal - <https://documentation.onesignal.com/docs>
7. Інтернет-ресурс, GithHub, GitHub Docs - <https://docs.github.com/en>
8. Хабібуллін І. Ш. Самовчитель XML. - СПб.: БХВ-Петербург, 2003. - 336 с.
9. Мак-Лахлін Б. Java та XML: Пер. з англ. - СПб.: Символ-Плюс, 2002. - 544 с.
10. Кей М. XSLT. Довідник програміста - СПб.: Символ-Плюс, 2002. - 1016 с.