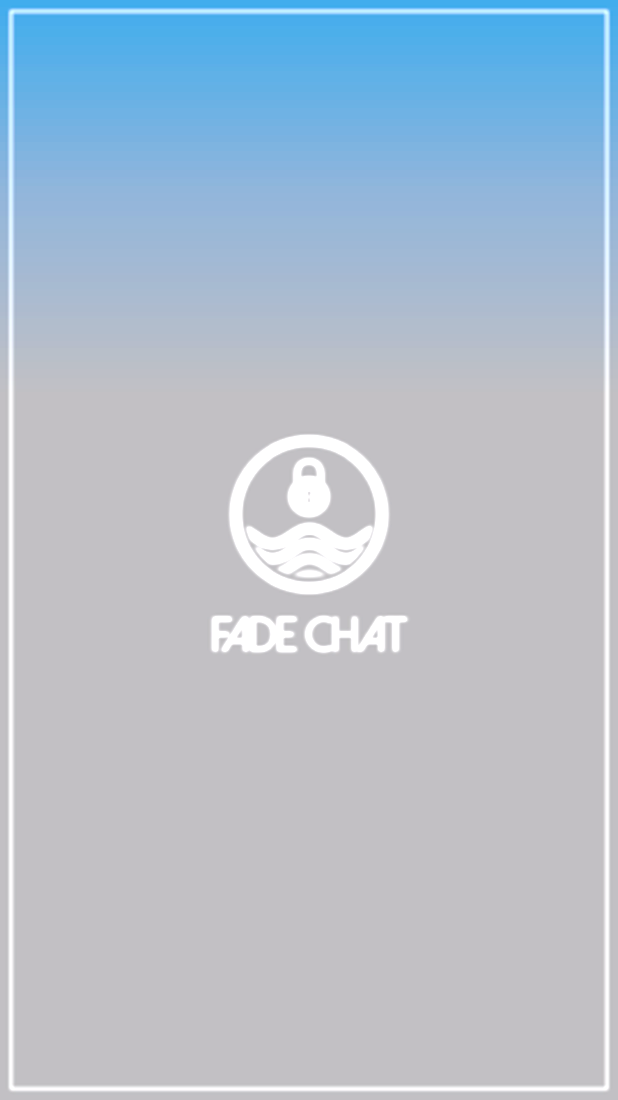
국민대학교 컴퓨터공학부

창의 소프트웨어 프로그램 1기 동계 훈련



|  |  |
| --- | --- |
| 프로젝트 명 | FadeChat |
| 문서 제목 | 프로젝트 계획서 |

|  |  |
| --- | --- |
| Date | 2015-MAR-08 |

|  |  |
| --- | --- |
| 팀원 | 20113458 김민규 |
| 20113304 이동욱 |
| 담당 교수 | 임 성수 교수님 |

# 프로젝트 목표

21세기 정보화시대가 시작되면서 현재까지 수많은 정보와 데이터들이 넘쳐나고 있다.

그로인해 “보안” 이라는 문제가 화두에 오르게 되었고 , “보안”을 이번 프로젝트의 키워드로 정하게 되었다. 그 중 현대인들이 가장 많이 이용하는 메신저에 접목시켜보기로 하였다.

앞서 ‘민중의소리’ 라는 언론사의 기사 일부를 발췌해보았다.

수사기관이 압수수색영장을 발부받아 카카오톡 대화내용을 가져가는 경우에 대해 국민들은 ‘실시간 대화’를 누군가 엿보고 있다고 생각하는 반면 수사기관들은 ‘저장된 자료’라며 법적으로 문제가 없다고 주장한다. 국민은 압수수색을 감청이라고 생각하고, 수사기관은 압수수색이라고 강변한다. 국민 감정과 법체계가 서로 충돌하고 있는 것이다.

이처럼 국민들은 자신의 사생활이 담긴 여러가지의 정보가 제 3자의 손으로 들어갈 수 있다는 불안감에 휩싸였다. 물론 문제 거리를 해소할 수 있는 단서가 될 수 있지만, 그것으로 인해 무관한 수많은 사람들의 정보가 유출된다는 것은 묵인할 수 있는 부분은 아니다.

그리하여 생각 해 낸 목표가 메신저 역할을 하지만 개개인의 소중한 정보를 보호 할 수 있도록 개인의 필요성에 의하여 중요한 대화의 일부분 혹은 전체정보를 삭제시킬수 있도록 하는것이 목표다.

# 프로젝트 계획

계획 순서

1. 서버와 네트워크 구상

* 프로토콜(MQTT ,AMQP, XMPP, etc)
* 서버(IBM WebSphere MQ, rabbitMQ,, mosquitto, etc)

1. Client 와 Server 간의 네트워크 연결

* Client 간의 메시지 통신
* Client 간의 대화 중 기호에 따라 상대방에게 내용을 남기거나 남기지 않는 기능을 구현.

1. UI 구축
2. 테스트 및 버그 수정

# 프로젝트 구현

1) 개발화경 설정

프로토콜: AMQP

서버 : RabbitMQ

AWS : EC2

Open Source : [RabbitMQ-Android-Chat](https://github.com/intrepidkarthi/RabbitMQ-Android-Chat) by [intrepidkarthi](https://github.com/intrepidkarthi)

(https://github.com/intrepidkarthi/RabbitMQ-Android-Chat)

Logo Design : 허광진 (광운대학교 3학년)

참고문헌 : 안드로이드 프로그래밍 정복 1,2 (김상형 저 , 한빛미디어)

2)소스코드 설명

- IntroActivity.java-

public class IntroActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.intro\_fadechat)//인트로 (로고)화면을 설정

Handler handler = new Handler();

handler.postDelayed(new Runnable() {

public void run() {

//intent 를 이용하여 현재 introActivity에서 MainActivity로 이동하는 객체생성

Intent intent = new Intent(IntroActivity.this, MainActivity.class);

//인텐트 실행(화면전환)

startActivity(intent);

finish();

}

}, 3000); //타이머를 지정 (3초로 지정)

}

}

IntroActivity는 Fadechat의 앰블럼을 보여주는 액티비티이다.

intro\_fadechat.xml 로 액티비티를 생성하였다.

핸들러를 이용하여 3초후 자동으로 intent하여 MainActivity로

넘어가도록 설정하였다.

-ServerInfo.java-

public class ServerInfo {

public static final String HOST="52.0.7.146"; // 서버주소

public static final int Port=5672; // AMQP 를 사용하는 포트주

public static final String ClientId="MK"; //User name 을 가질 String type 의 ID

public static final String UserName="DM"; //Rabbitmq 의 username

public static final String PassWord="DM"; //Rabbitmq 의 password

public static String EXCHANGE="room1"; //Server Exchange 이름

public static String Queue="mkQ2"; //Server Queue 이름

}

서버를 이용한 Chatting App 이기 때문에 서버연동에 필요한 기본적인 요소를 모았다.

그 내용으로는 AMQP 메시지 브로커 소프트웨어인 RabbitMQ의 호스트, 그리고 우리가 사용할 포트번호, 서버를 연동한 다중 통신이가능하지만 테스트 용으로 설정한 임의의 ClientID, RabbitMQ를 접속하기위한 아이디와 패스워드, 그리고 Exchange 와 Queue이름을 지정하였다.

-MsgAdapter.java-

public class MsgAdapter extends ArrayAdapter<Msg> {

private int resourceId;

public MsgAdapter(Context context, int textViewResourceId,

List<Msg> objects) {

super(context, textViewResourceId, objects);

resourceId = textViewResourceId;

}

Msg를 Parameter로 갖는 ArrayAdapter를 상속받아 context,

그리고 resourceID를 받았다.

@Override

public View getView(int position, View convertView, ViewGroup parent) {

Msg msg = getItem(position);

View view;

ViewHolder viewHolder;

…

또한 기존의 getView메서드를 오버라이드 하여 Msg객체를 생성하고 화면에 보여주기위하여 View , ViewHolder객체를 생성하였다.

그아래에는 case by case 에따라 (메시지를 receive 혹은 send할 때) 기존에 설정해 놓았던 채팅에 쓰일 물풍선, 타이머 그리고 fade 기능을 구현할때의 물풍선, 타이머 등을 일일이 지정하였다. 소스코드가 길어 내용은 이하생략하였다.

-Msg.java-

public class Msg {

public String exchange = "";

public String routingKey = "";

public static final int TYPE\_RECEIVED = 0; //final로 메세지를 받을때는 0으로 설정

public static final int TYPE\_SENT=1; //final로 메세지를 보낼때는 1로 설정

private String content; //내용을 가져올 변수

private int type;

private int timer; //fade기능을 위해 제한시간 기능을 넣은 변수

public Msg(){ //기본 메시지는 시간을 0으로 지정

this.timer =0;

}

public Msg(String routingKey, String content) {

this.routingKey = routingKey;

this.content = content;

}

public Msg(String exchange, String routingKey, String content) {

this.exchange = exchange;

this.routingKey = routingKey;

this.content = content;

}

//timer의 getter setter 메서드

public int getTimer() {

return timer;

}

public void setTimer(int timer) {

this.timer = timer;

}

public String toString() {

if (exchange.length() > 0)

{

return String.format("Exchange='%s', Key='%s', '%s'", exchange,

routingKey, content);

}

else

{

return String.format("Key='%s', '%s'", routingKey, content);

}

}

//내용을 갖고오는 컨텐트 getter 메서드

public String getContent(){

return content;

}

//type을 갖고오는 getter, setter 메서드

public void setType(int type) {

this.type = type;

}

public int getType(){

return type;

}

}

Msg 클래스는 기존의 Exchange , Routing Key ,Content(메시지내용), Fade기능을 위한 Timer를 가지고와 그것에대한 getter, setter 메서드를 제공해준다.

-LoginActivity.java-

public class LoginActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.login\_fadechat);

//Button

Button btnCall = (Button)findViewById(R.id.sendButton);

btnCall.setOnClickListener(new Button.OnClickListener() {

//start 버튼 클릭시

public void onClick(View v) {

EditText userName = (EditText) findViewById(R.id.editText1);

EditText roomName = (EditText) findViewById(R.id.editText2);

//아이디 , 방정보, 큐 이름 입력받기.

ServerInfo.ClientId=userName.getText().toString();

ServerInfo.EXCHANGE=roomName.getText().toString();

ServerInfo.Queue=ServerInfo.ClientId;

//클라이언트아이디 와 방이름(exchange)가 입력되지않았을시 메세지를 보냄.

if("".equals(ServerInfo.ClientId) || "".equals(ServerInfo.EXCHANGE))

{

Toast.makeText(LoginActivity.this, "You must fill the blank",

Toast.LENGTH\_LONG).show();

}

else

{

//둘다 입력이 제대로 들어왔을시 화면전환

Intent intent = new Intent(LoginActivity.this, MainActivity.class);

startActivity(intent);

}

} });

}}

버튼을 클릭시 액티비티에 지정된 사용자이름 입력 EditText와

룸이름 EditText를 받아온후 그 두 이름이 두개 모두 입력시 되어있을 시

메인 엑티비티로 intent시키도록 하였다.

-MainActivity.java-

public class MainActivity extends Activity {

…

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

new Recv2().execute();

setContentView(R.layout.activity\_main); //메인엑티비티 레이아웃을 불러드림

adapter = new MsgAdapter(MainActivity.this, R.layout.msg\_item, msgList);

Toast.makeText(MainActivity.this, "Copyright(c) 동욱,민규 All rights reserved!",

Toast.LENGTH\_LONG).show();

MsgRemover remover=new MsgRemover(); // fade메세지 삭제를 위해 MsgRemover객체 생성

// MsgRemover에 있는 OnRemoveMessageHandler에 있는 interface 구현

remover.setOnRemoveMessageHandler(new OnRemoveMessageHandler() {

@Override

public void OnRemoveMessage(List<Msg> rMsgs) {

// TODO Auto-generated method stub

for(Msg msg:rMsgs){

msgList.remove(msg); // 메세지 삭제

adapter.notifyDataSetChanged();

msgListView.setSelection(msgList.size());

}

}

@Override // 제한시간표시

public void OnRemoveTimer() {

// TODO Auto-generated method stub

adapter.notifyDataSetChanged();

msgListView.setSelection(msgList.size());

}

});

remover.start(); // Remover 실

inputText = (EditText) findViewById(R.id.input\_text);

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

msgListView = (ListView) findViewById(R.id.msg\_list\_view);

msgListView.setAdapter(adapter);

toggle = (ToggleButton) findViewById(R.id.toggleButton1);

send.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

String content = inputText.getText().toString();

if (!"".equals(content)) {

int timer=0; // 처음은 not fade로 설

// 메세지 생성

Msg msg = new Msg(ServerInfo.EXCHANGE,"",content);

//타입은 sent 로 잡는다. (메세지 보내기)

msg.setType(Msg.TYPE\_SENT);

// fade가 선택되었을때 타이머를 9초로 설정

if(toggle.isChecked())

timer=9;

// 메세지에 시간적용

msg.setTimer(timer);

// 방이름과 큐 정보를 보내고 ( 여러개 방 생성했을 때를 고려하여), 메세지에는 아이디,메세지내용,시간을 보낸다.

new Send(ServerInfo.EXCHANGE,ServerInfo.Queue).execute(ServerInfo.ClientId+"///"+content+"///"+timer);

msgList.add(msg); //메세지 추가

adapter.notifyDataSetChanged();

msgListView.setSelection(msgList.size());

inputText.setText("");

}

}

});

// 메세지가 왔을때 ui에 적용하기위해 consumer객체에 interface 내부에 메세드 구현

consumer.setOnReceiveMessageHandler(new OnReceiveMessageHandler() {

@Override

public void onReceiveMessage(byte[] message) {

// TODO Auto-generated method stub

String text = "";

try {

text = new String(message, "UTF8"); // message를 유니코드로 변경 (byte 단위로 왔음으로)

// Tokenizer를 생성하여 "///" 로 아이디, 메세지, 시간을 구분한다.

StringTokenizer tokenizer = new StringTokenizer(text, "///");

String id=tokenizer.nextToken();

String word=tokenizer.nextToken();

int timer= Integer.parseInt(tokenizer.nextToken());

// fade 가 아닌 메세지 갯수 추가 (remover 사용을 위해)

if(timer==0)

not\_fadeNumber++;

// 받아온 메세지가 내 아이디랑 같으면 추가하지 않고 다르면 추

if(!id.equals(ServerInfo.ClientId))

{

Msg msg = new Msg(ServerInfo.EXCHANGE,"",id+":"+word);

msg.setType(Msg.TYPE\_RECEIVED);

msg.setTimer(timer);

msgList.add(msg); //메세지 추가

adapter.notifyDataSetChanged();

msgListView.setSelection(msgList.size());

}

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

});

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

// Handle action bar item clicks here. The action bar will

// automatically handle clicks on the Home/Up button, so long

// as you specify a parent activity in AndroidManifest.xml.

int id = item.getItemId();

if (id == R.id.action\_settings) {

return true;

}

return super.onOptionsItemSelected(item);

}

private class Recv2 extends AsyncTask<String, Void, Void> {

@Override

protected Void doInBackground(String... Message) {

try {

// Connect to broker

consumer.connectServer();

consumer.start();

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

// TODO Auto-generated method stub

return null;

}

}

@Override

public boolean onKeyDown(int keyCode, KeyEvent event) {

switch (keyCode) {

case KeyEvent.KEYCODE\_BACK:

consumer.dispose();

this.moveTaskToBack(true);

this.finish();

android.os.Process.killProcess(android.os.Process.myPid());

System.exit(0);

return true;

}

return super.onKeyDown(keyCode, event);

}

}

-consumer.java-

public class Consumer {

private Connection mConnection; // AMQP를 이용하여 서버에 설치되어있는 RabbitMQ에 Connection 연결

private Channel mChannel; // AMQP를 이용하여 서버에 설치되어있는 RabbitMQ에 Channel 연결

private Consumer(){};

public Channel getChannel()

{

return mChannel;

}

private QueueingConsumer mConsumer; // Queue를 사용하는 Consumer

private boolean running; // 메세지를 받아오는 thread의 flag

private Handler mConsumeHandler = new Handler(); // 메세지를 받아오는 쓰레드를 실행시키기위한 핸들러

private Handler mMessageHandler = new Handler(); // 쓰레드가 메세지를 받아오면 UI변경하기위한 핸들러

public static Consumer consumer=new Consumer(); // 싱글톤 디자인으로 프로그램 consumer를 한개로 만든다.

public static Consumer getConsumer(){

if(consumer==null)

consumer=new Consumer();

return consumer;

}

private byte[] mLastMessage; // 최근에 받아온 메세지

public interface OnReceiveMessageHandler{ // 쓰레드가 메세지를 받아오면 핸들러가 실행할 메세드를 정의한 인터페이스

public void onReceiveMessage(byte[] message);

};

private OnReceiveMessageHandler mOnReceiveMessageHandler=null;

// 인터페이스의 메세드를 구현한 것을 할당

public void setOnReceiveMessageHandler(OnReceiveMessageHandler handler){

mOnReceiveMessageHandler = handler;

};

// 메세지가왔을때 핸들러에 보낼 쓰레드

final Runnable mReturnMessage = new Runnable() {

public void run() {

mOnReceiveMessageHandler.onReceiveMessage(mLastMessage);

}

};

// 프로그램이 시작했을때 핸들러에보낼 메서드

final Runnable mConsumeRunner = new Runnable() {

public void run() {

Consume();

}

};

// 서버에 연결

public boolean connectServer(){

if(mConnection!= null && mChannel.isOpen() )

return true;

try

{

ConnectionFactory factory = new ConnectionFactory();

factory.setHost(ServerInfo.HOST);

// 서버에 연결할 아이디, 비번 설정

factory.setUsername(ServerInfo.UserName);

factory.setPassword(ServerInfo.PassWord);

// connection,channel 연

mConnection = factory.newConnection();

mChannel = mConnection.createChannel();

// "fanout"방식에 채팅방 생성

mChannel.exchangeDeclare(ServerInfo.EXCHANGE, "fanout", true);

return true;

}

catch (Exception e)

{

e.printStackTrace();

return false;

}

}

public void start()

{

//서버에 연결되었으면 바인딩

try {

// 큐 생성

mChannel.queueDeclare(ServerInfo.Queue, false, false, false, null);

// consumer 생성 및 channel연결

if(mChannel.getDefaultConsumer()==null)

{

mConsumer= new QueueingConsumer(mChannel);

mChannel.basicConsume(ServerInfo.Queue, true, mConsumer);

}

} catch (IOException e) {

e.printStackTrace();

}

try {

//큐바인딩

mChannel.queueBind(ServerInfo.Queue,ServerInfo.EXCHANGE, "");

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// 메세지를 받아오는 쓰레드 실행을 위해 핸들러에 쓰레드를 보내줌

running = true;

mConsumeHandler.post(mConsumeRunner);

}

// 메세지를 받아오는 쓰레드를 실행하는 메세드

private void Consume()

{

Thread thread = new Thread()

{

@Override

public void run() {

while(running){

QueueingConsumer.Delivery delivery;

try {

delivery = mConsumer.nextDelivery();

mLastMessage = delivery.getBody();

// 새로운 메세지가 오면 핸들러에 쓰레드를 보내서 UI변경

mMessageHandler.post(mReturnMessage);

} catch (InterruptedException ie) {

ie.printStackTrace();

}

}

}

};

thread.start();

}

// 연결종료

public void dispose()

{

this.running = false;

try {

if (mConnection!=null)

mConnection.close();

if (mChannel != null)

mChannel.abort();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

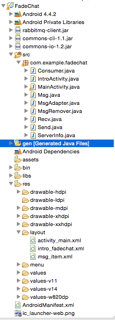
}

}

Consumer 클래스는 서버와 연결하는 역할을 하며, rabbitmq의 consumer를 생성하고, 쓰레드를 통해 메세지를 받아오고 핸들러를 이용하여 유아이를 변경하는 역할을 합니다.

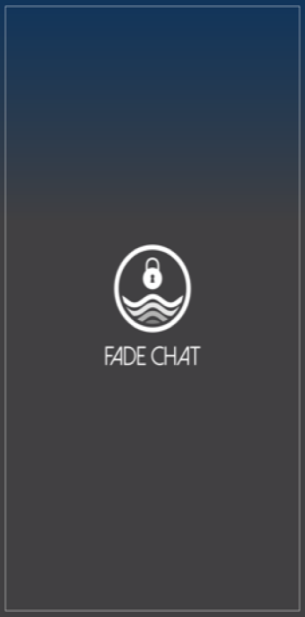
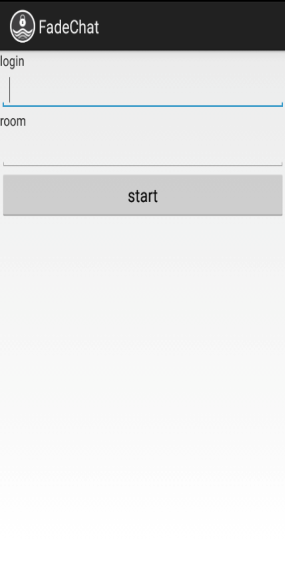
LoginActivity를 통해서 exchange(방)와 name(clientID)를 받아와서 채팅방 생성 및 큐를 바인딩하고 큐를 사용하는 컨슈머를 바인딩합니다. 프로그램이 실행되는 동안 쓰레드가 계속 돌도록하여서 메세지를 받아오고 프로그램 종료시 dispose()를 통해서 연결을 종료합니다.

3)프로그램 구성 파일



9개의 java 파일과 3개의 activity를 구성하는 xml파일과

AndroidManifest.xml파일로 구성

인트로 채팅배경로그인

앱 아이콘

fade off(receive) fade on(receive)

fade off(send) fade on(receive)

메시지 창 (9-patch 하기전의 아이콘이라 위처럼 선이나와있음)

 fade mode 에서 남은 시간을 알려주기위한 timer background

# 향후 숙제

현재로서는 데이터베이스를 연동하지 않아서 프로그램을 종료하였을때, 방에 있던 채팅정보가 다 날아가는 단점이 있다. 또한, 푸쉬알림을 사용하지 않아서, 프로그램을 실행하였을 때에만 메세지를 수신할 수 밖에 없는 단점이 있다.

향후 이러한 문제점들을 보완하여서 어플을 출시하고 싶다.

# 느낀 점

-이동욱,김민규 (컴퓨터공학부 20113304,20113258) –

이번 프로젝트를 하면서 느낀 점은 상당하다. 어플리케이션 개발에 있어서 처음부터 끝까지 어느 하나 쉬운 부분이 없다는 것을 느낄 수 있었다. 반면에 어플리케이션(이하 앱) 개발이 상당히 흥미로운 작업이라는 것 또한 느꼈다.

2명이서 같이 프로젝트를 진행하기로 한 후 , 어떠한 안드로이드 기반의 앱을 만들지에 대해 상의를 했다. 이것이 가장 흥미로웠던 시간이 아니었나 싶다. 개발자로서 사용자 및 소비자들의 needs를 충족 시킬수 있는 앱이 어떤것일까.. 둘이 이 주제로 하루 내내 고민을 하였다. 현 시대의 트렌드를 알고있어야하고, 기존의 앱보다 참신해야했다. 손으로 적어가며 여러가지의 아이디어를 내놓았지만 실제로는 마켓에 존재하는 앱이 대부분이었으며, 상당 유사한 앱들도 많이 있었다. 고심의 끝에 IT관련 기사들을 쭉 읽어보고 “보안”이라는 키워드에 중점을 두기로하였다. 사용자들의 수많은 데이터들이 존재하는 이 시대에 그 데이터들은 생각외로 잘 돌아나니고있었다. 위에서 목표에서도 언급한 바 있지만, 카카오톡이라는 회사의 채팅프로그램이 개인정보 유출로 인하여 사용자들로 하여금 충격을 주었다. 그리하여 우리는 정보를 보안할 수 있는 색다른 채팅 앱을 만들어보고싶었다. 물론 두명이서 한달이라는 기간내에 우리가 생각한 아이디어를 완벽히 구현하기에는 한계가있다고 생각하였다. 하지만 그에 따라 방향을 정하고 어느 정도까지 만들 수 있다고 생각을 하였기에 Fadechat 이라는 어플을 만들게 되었다.

아이디어가 앱 만드는데 있어 반 이상의 역할을 했다해도 과언이 아닌 것 같다.

그리하여 우리는 안드로이드 기반의 앱을 만들고자 Server-Client , UI , 등등 다소 생소한 부분을 접하게 되었고 그에 따라 수많은 시행착오를 하게되었다.

어떠한 프로토콜을 쓸까 부터 마지막 UI design 까지 아마 사실상 현재 앱에 필요치 않은 시간들도 많이 할애하였다. 물론 지나고나서 생각하니 그 모든것들이 개발자한테 있어 피가되고 살이되는 과정이라고 생각한다.

아직 많은 버그들이 존재하고, 불안정하지만 우리가 만든 앱으로 대화를 하며 만들고자 했던 기능이 실행될때의 감정은 컴퓨터를 전공하면서 처음으로 느꼈던 것 같다.

부족하지만 밀어주고 당겨준 서로에게 너무 고마운 시간이었다.

아이디어의 중요성, 과정의 중요성 그리고 그 안의 협업 즉 팀원간의 소통의 중요성을 절실히 느낄 수 있었던 프로젝트였던 것 같다.