Development of a WebRTC enhanced mobile solution

Case study

LogMeIn Rescue - Balazs S Banyai

Intro

- Who?
  - Balazs S Banyai - Senior Software Engineer, Inventor of Rescue Lens
- What?
  - Rescue Lens - An app that brings a brand new approach to the remote support industry
- Why?
  - WebRTC is an awesome piece of technology that is spreading but it still has it’s own pitfalls
  - We’ll also enumerate the rest of the technologies we used during development

Lens in a nutshell

https://www.logmeinrescue.com/features/lens

- Requirement
  - An app that is capable of sending live camera stream to a PC over the internet
- Scenario
  - End user has an issue with his motorbike
  - He downloads the Lens App as instructed by the vendor
  - A support session established between a field expert and the end user
  - The expert can see the issue using the live stream, and can advise on fixing the issue
  - The expert can also draw on the camera image, which will be shown on the end user’s screen

Lens in a nutshell

https://www.logmeinrescue.com/features/lens

Lens in a nutshell

https://www.logmeinrescue.com/features/lens

- The prototype was developed on the company Hackathon in one week
- It took a few months to make it production ready

Why WebRTC?

http://www.webrtc.org/

- A library that provides **R**eal**T**ime**C**ommunication
  - Audio
  - Video
  - And much more on the Data Channel
- Adaptive bandwidth management
- Supported natively by major browsers (Chrome, FF)
- Mature enough that Hangouts uses it
- Thin API, seemingly easy to use

--> Which makes it an excellent tool for the task <--

Why WebRTC?

http://iswebrtcreadyyet.com/

- It requires a channel between the parties
- This channel is used to exchange messages (Signaling)
- After signaling, the media stream will run through the channel established by WebRTC
  - P2P by default
  - requires STUN servers in order to discover the public IP
  - uses TURN servers if P2P not available

WebRTC challenges - building

http://www.webrtc.org/

- It took 2 days just to build the project
- Various custom built tools and scripts
- Huge repository
- Android only builds on linux machine
- Native lib with java wrapper

WebRTC challenges - device list

http://www.webrtc.org/

- Completely takes control of the standard Camera API
- Device & resolution is set as a String!
- The camera cannot be controlled at all
  - No flash control
  - No autofocus control
  - ...

WebRTC challenges - device rotation

http://www.webrtc.org/

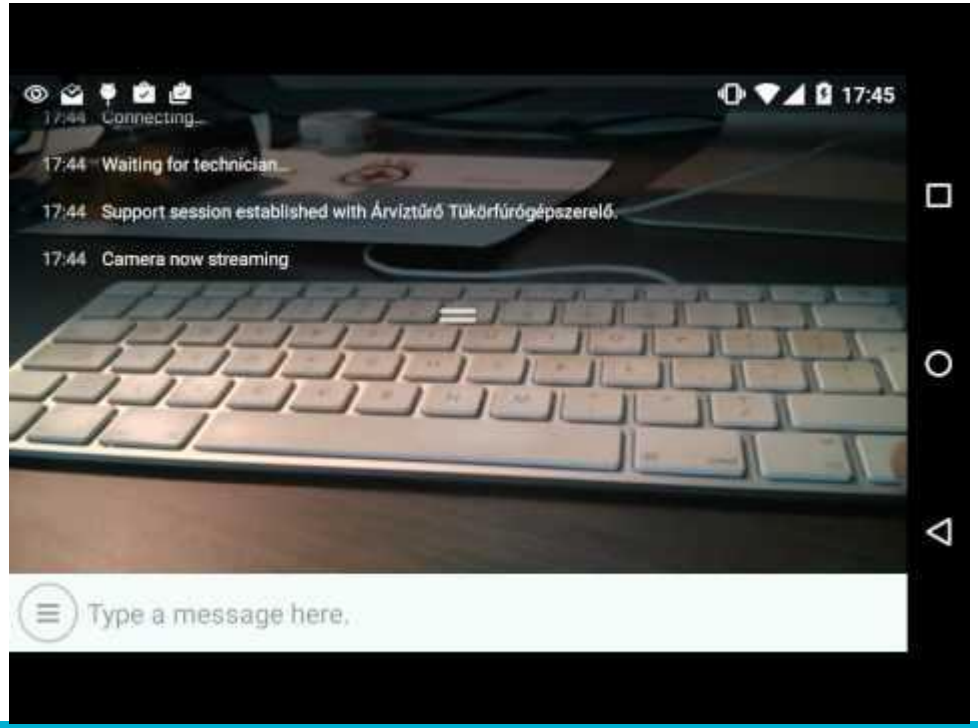
- Since the camera object is owned by the lib, there is no control over the camera image rotation
- The lib rotates the image automatically
- This feature cannot be turned off

Effect:

- Before the configChange would change the layout, the camera image is rotated immediately by the lib

WebRTC challenges - device rotation

http://www.webrtc.org/



OpenCV optical flow

http://opencv.org/

- OpenCV
  - Extensive set of advanced image processing algorithms
  - Distributed as a separate APK (OpenCV manager)
  - Used as a remote service by apps
- We use it to keep the drawing in-place on the camera image when the device is moved

OpenCV optical flow

http://opencv.org/

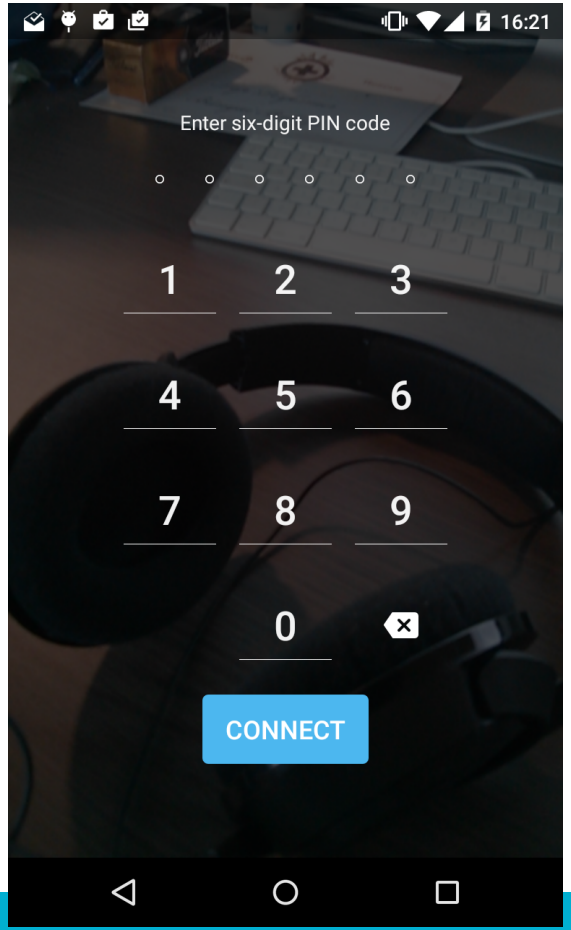
OpenCV optical flow

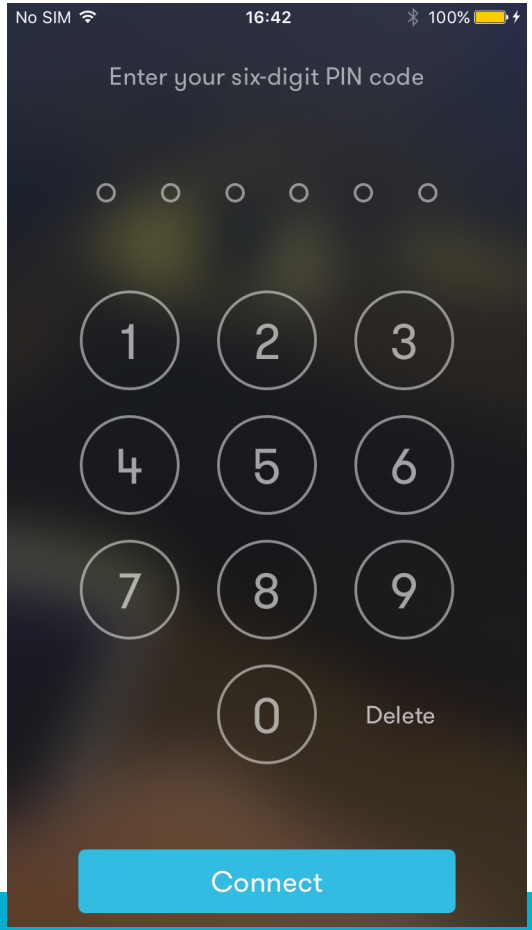
http://opencv.org/

- Algorithm
  - got first frame, detect feature points (FP) and store them
  - got new frame, detect FPs and compare to previous state
  - calculate average translation from the result vectors
  - store the FPs from the current image
- Prototype implemented on Android
- Later this feature was moved to the technician’s side (video streaming and image processing simultaneously was just too much for the mobile device)

OpenGL blur

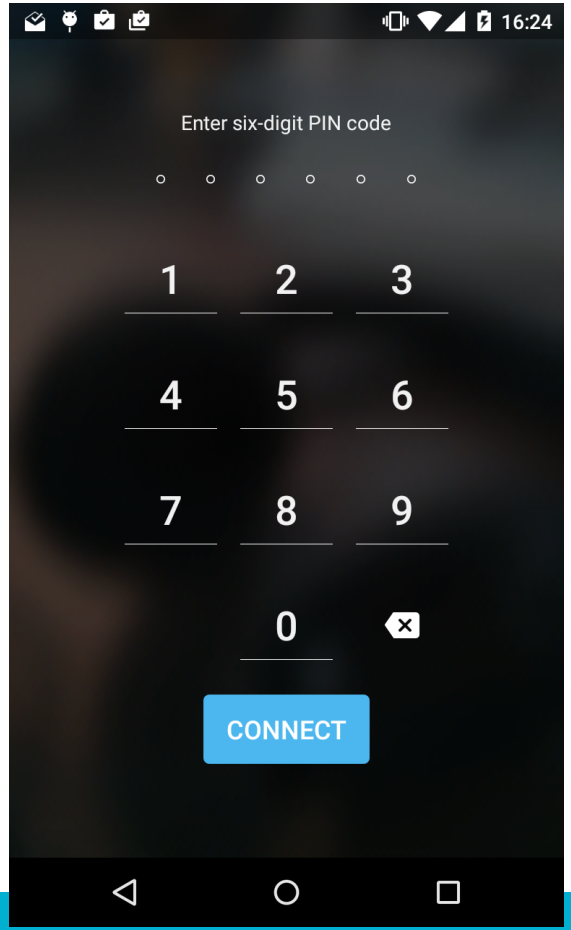
- Main screens iOS vs Android
- Background is camera stream
- Note the awesome blur effect

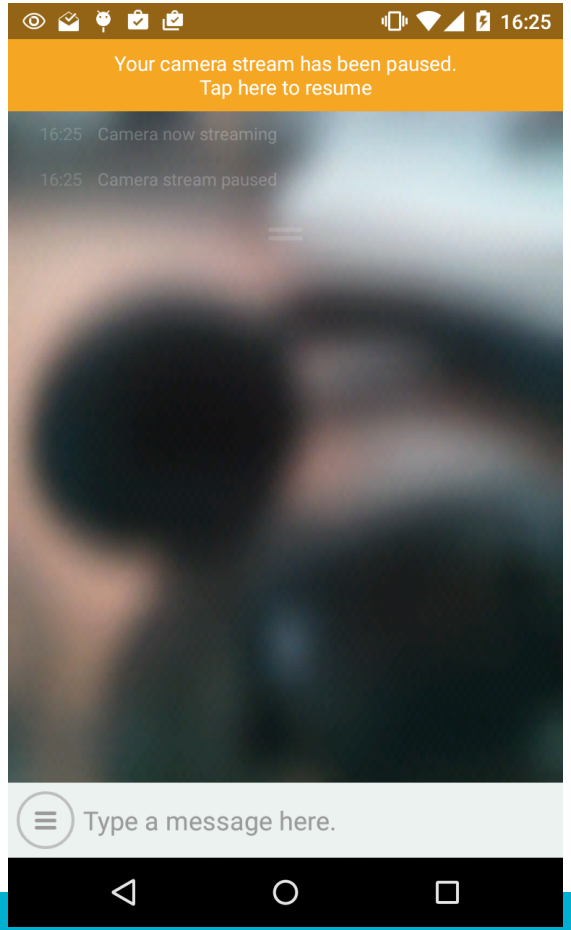




OpenGL blur

- Realtime blur
- Implemented using shaders
- Runs on GPU
  - Fast
  - No additional load on CPU





Guice

https://github.com/google/guice

- Powerful IoC container
- Promotes constructor injection
- Simplifies unit testing
- Easy configuration
- Negative: introduces runtime overhead

ButterKnife

http://jakewharton.github.io/butterknife/

- View injection lib
- Helps to get rid of the superfluous *findViewById()* syntax
- Type safety
  - No more casting

ButterKnife

http://jakewharton.github.io/butterknife/

```
public class MainActivity extends Activity {

 Button button;

 @Override
 protected void onCreate(Bundle b) {
  super.onCreate(b);
  setContentView(R.layout.activity_main);

  button = (Button)findViewById(R.id.submitButton);
 }
}
```

```
public class MainActivity extends Activity {

 @Bind(R.id.submitButton)
 Button button;

 @Override
 protected void onCreate(Bundle b) {
  super.onCreate(b);
  setContentView(R.layout.activity_main);

  ButterKnife.bind(this);
 }
}
```

ButterKnife

http://jakewharton.github.io/butterknife/

- No more anonymous listeners
- Compile time code generation means zero runtime overhead

ButterKnife

http://jakewharton.github.io/butterknife/

```
...
final Button button = (Button) findViewById(R.id.
submitButton);
button.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       Log.d(TAG, button.getText().toString());
   }
});

...
```

```
public class MainActivity extends Activity {

 @OnClick(R.id.submitButton)
 public void buttonClicked(Button button) {
   Log.d(TAG, button.getText().toString())
 }
}
```

EventBus

https://github.com/google/guava/wiki/EventBusExplained

- Pub-Sub pattern
- Usually global eventBus instance
- Subscribers need to *register()*
- Publishers can post to the bus
- Everyone who listens to *Event* will be notified
  (if Event or any subclass dispatched)

EventBus

https://github.com/google/guava/wiki/EventBusExplained

- Link to prezi
- Example
  - Notification is only interested in the connection active/inactive status
  - → So it only subscribes to the related event!
  - Implementing an interface would require the client to handle all the callbacks

  - Detailed status can be accessed by subscribing to more detailed event types

EventBus

https://github.com/google/guava/wiki/EventBusExplained

```
class ChangeRecorder {
     void setCustomer(Customer cust) {
       cust.addChangeListener(new ChangeListener() {
         public void customerChanged(ChangeEvent e) {
           recordChange(e.getChange());
         }
       };
     }
   }
```

```
// Class is typically registered by the container.
class EventBusChangeRecorder {
    @Subscribe
    public void recordCustomerChange(ChangeEvent e) {
        recordChange(e.getChange());
    }
}
```

EventBus

https://github.com/google/guava/wiki/EventBusExplained

- Loose coupling
- No monster switch-cases
- Let client decide the granularity that they are interested in

Java dynamic proxies

http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Proxy.html

---

- Extremely handy tools to decorate interfaces
- Implementing interface(s) at runtime
- Maps all method invocations to an *InvocationHandler*
- The implementation can decide what to do with the invocation
  - Post it to another thread, to make a library thread safe?
  - Log all the invocations?
  - Implement caching mechanism?
  - ...

Java dynamic proxies

http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Proxy.html

```
final List list = new ArrayList();
ClassLoader loader = getClass().getClassLoader();
Class<?>[] interfaces = { List.class };
InvocationHandler handler = new InvocationHandler() {
   @Override
   public Object invoke(Object o, Method method, Object[] objects) throws Throwable {
       Log.d(TAG, method.getName() + "invoked");
       return method.invoke(list, objects);
   }
};
List loggedList = (List) Proxy.newProxyInstance(loader, interfaces, handler);


loggedList.clear(); // Prints “clear invoked” then clears the list
```

Q & A