Insight Security
Richard Basdeo
Merna Eldesouki
Eric Benjamin
Final Transcript

Slide 1:
Welcome to our Insight Security presentation, I'm Richard.
I'm Merna.
And I'm Eric! Welcome to our final demo presentation.

Slide 2:

Richard: Insight Security is our home security app. It gives life to old devices by turning them into cameras and noise detectors. Traditionally, security systems cost around $200 and require a yearly membership. Our app is a more cost-efficient solution to allow people from all backgrounds to have a security system.

We provide security in two ways - the first way is taking a picture every 30 minutes, the second way is via sound by listening to the decibels in the room and notifying the user on another device if there is a noise spike.

Slide 3:

Merna: We realized trying to show you our app by screensharing an emulator would be tricky so instead we added screen recordings instead to demonstrate the app's functionality

So the first video demonstrates the camera mode where the app takes pictures and listens for noise spikes *plays video*. Once you launch the app, you log into your Google account. You're prompted to agree to a disclaimer. Then you choose which mode you want to go into, in this case camera mode. The app listens to fluctuations in noise and once a noise spike is detected (through an algorithm we discuss later), the app automatically takes a picture and notifies the user through a notification. Alternatively, you can start the timer to take a picture every 30 minutes (although for demo purposes we have it set to 30 seconds). The pictures get uploaded to the database for the user to view on their phone.

We have a separate recording for the viewer mode *plays video*. In viewer mode, the pictures that the "camera" device has taken are available for the user to view on any phone they are signed into. Each picture has the date and time it was taken displayed with it.

Eric and I are signed into the same Google account, so now I will create a noise spike on my phone and Eric will show you the notification he gets on his phone.

Slide 4:

Eric: Since our MVP demo we have added some new features to our app. This includes a noise event algorithm where the app compares the current noise level to the previous one. If it is over a certain level and it is a percentage higher than the previously recorded level, a noise event takes place.

Another feature we added is notification protection. To prevent the app from spamming the user we implemented a cloud function which prevents a user from getting notified more than once every five minutes.

A third feature is the integrity agreement in which the user has to agree to use the app for its intended use to make sure the app is used ethically. If the user declines, they are logged out of the app.

Lastly, we implemented a battery saving feature where the screen turns off once it is in camera mode and records noise levels to conserve battery.

Slide 5:

Richard: A major feature we added is image capture. Once a noise spike is detected, a picture is taken. In addition, the user can choose to start a timer which takes a picture once every 30 minutes regardless of whether a noise spike occurs. The pictures are then available to the user through viewer mode which they can access from any phone as long as they're signed into the app with the same account.

Slide 6:
Richard: We also added a file path creation feature so that each image would get a timestamp, allowing the app to automatically upload photos and for the user to delete old photos.

Slide 7:

Merna: As for the tools we used, we wrote our app using Java. We used the Google API for the Google sign in feature. We used Firebase for our database as well for sending out notifications to users. As for NodeJS we used that for our Firebase cloud functions since that is the only way to execute the cloud functions.

Slide 8:

Eric: The way we split up the work was that I worked on the notifications as well as managing the database, Merna was responsible for designing the UI and implementing the noise

detection, and Richard worked on the image capture, Google sign-in and managing the database

Slide 9:

Eric: One challenge we faced was trying to prevent the app from sending too many notifications (as you can see we had an issue where the app would spam the user with notifications)

Richard: Another challenge was trying to configure the app so that it would take pictures without needing the user to manually take them

Merna: And we also had issues adjusting the noise event algorithm so that it wouldn't send false-positives or ignore major noise spikes.

Slide 10:

Merna: In the future, some features we hope to add are motion detection to go along with noise detection to make our app a more robust security app, recording video after a noise spike is detected which would also be available for the user to view from their phone, and starting a livestream so that a user could check in on their home while they are away.

Slide 11:

Merna: That concludes the first part of our presentation. Any questions?

Slide 12:

Eric: For our first tech talk, Merna and I will discuss Firebase cloud functions.

Slide 13:

Eric: This presentation is about Firebase Cloud Functions, but I wanted to give you a bit of background on Firebase first to put things into context. However, I'll be very brief about it. **Firebase acts as your server, API and Datastore.** That means all you have to do when developing your app is send your data to Firebase and query it back. You don't have to worry about managing servers or implementing APIs. Firebase handles all of these things, which is why it's considered a Backend as a Service. Firebase is great and you should all investigate it if you haven't already.

Slide 14:

Eric: Now, for the subject of this presentation - Firebase Cloud Functions. **Cloud Functions automatically run your code in response to events in your Firebase database.** Basically, you write the code in JavaScript in TypeScript, and then tell Firebase when you want it to trigger. Let me give you an example from this class. Let's say the Codenames team wants their app to be family-friendly and are worried about users typing inappropriate language in the chat. They would write a cloud function that parses a string and looks for prohibited words. They would set that function to trigger when data is written to the path in their Firebase database that holds their in-game chat. Now, when a user types something, Firebase will see that data has been written in that path and run the function on the data, finding any inappropriate language.

When you utilize Cloud Functions, Firebase will handle storing and running your code on Google's servers, which you don't have to worry about managing. It'll automatically provision and decommission servers to match your usage levels. It'll also insulate your code from users' devices to keep it private and tamper-proof.

Slide 15:

Eric:Firebase Cloud Functions are extremely useful and have an infinite number of use cases. As another example, I'll talk a bit about how our project utilized them. One of our goals was to notify users via push notifications when a device linked to their account detected a surprising level of noise. Firebase has another great feature called Firebase Cloud Messaging. This tool sends push notifications to groups of users. The problem we faced was that Firebase Cloud Messaging generally is triggered manually. So, how did we end up making it automatically notify users of noise events? Firebase Cloud Functions!

When a device detected a noise event, it would set the noiseEvent value to true in the database for that user. The cloud function would trigger from that data being updated, and it would utilize Firebase Cloud Messaging to send notifications to users based on the original user's data. This is a great example of how Firebase Cloud Functions integrates with all of Firebase's useful tools to meet a large variety of requirements.

Slide 16:

Eric:**For mobile apps, the main alternative to Google Cloud Functions is AppEngine.** We wanted to talk a little bit about the limitations of Firebase Cloud Functions in relation to AppExchange instances, so that you can make informed choices for your future projects.

Firebase's convenient management of your instances and scaling can be limiting, as there is no way to influence these manually. Firebase will decide how to allocate resources to your project based on your user's usage patterns.

Firebase Cloud Functions are limited to nine minutes running time. They're meant for lightweight, simple computations -- anything more than that will get expensive real fast, as

Merna will be going over in the next slide. AppEngine has no such restriction, and is better in general for executing complex algorithms.

Slide 17:

Merna: We've built a costs comparison chart between Firebase Cloud Functions and Google AppEngine. Each service grants a limited amount of free usage, and then charges you relative to how much you use their service. I know this is kind of hard to eyeball just looking at the chart, but Firebase Cloud Functions offer a lot less free usage, and can be a bit more expensive than AppEngine.

Slide 18:

Merna: If Firebase Cloud Functions isn't always the cheapest option, why do people use it? One reason is that Firebase is extremely easy to set up and use. After your project is initialized, you can simply type 'firebase deploy' to deploy the latest version of your cloud function to your Firebase project. Easy as that!
Another reason for Firebase's success is its amazing documentation. They also have a Github repository with over forty sample projects. These projects demonstrate a huge variety of use cases, from common requirements like user authentication to integrations with third party tools such as LinkedIn and Paypal.

Slide 19:

Merna: Access the Firebase console, and click "Add project", or select an existing Google Cloud Platform project. Then, follow the steps on your screen until you can click the "Create project" button!

Slide 20:

Merna: The next step is to install NodeJS, and then install the Firebase Command Line Interface in any way you'd like. We've found that the easiest way is to install it via the command line using the command displayed.

Slide 21:

Merna: Next, you'll initialize your project. Initializing Firebase SDK connects your JavaScript or TypeScript code with your Firebase Project.

To do this, you'll run 'firebase login' in your command line, which will open your browser and let you connect to the Firebase Project you configured earlier. Then, you'll run 'firebase init functions', and follow the steps to finish initializing your project.

Slide 22:

Merna: After you've written or updated your cloud function, all you have to do to deploy it to your Firebase project is use the 'firebase deploy' command! At that point, your cloud function will be active and ready to automatically run.

Slide 23:

Merna: We hope that setting up Firebase looked as painless as it actually was, and that you'll consider it for your own projects. Thank you for watching our presentation!

Slide 24:
Richard:
Android was originally using camera api but it got deprecated roughly 5 years ago. Camera2 api was introduced and this is pretty much an image comparison between both of them. Camera api was mostly meant for pointing and shooting to take an image with limited customization. However Camera2 api is like a DSLR camera with a lot more features to play around with giving you more control of the camera and how the images come out.

Slide 25:
Richard:
Camera2 api was built with pipelining in mind. A little refresher on pipelining is the laundry example. Let's say you have multiple loads of laundry to do. You can do your laundry using the method in the upper picture by completing the full process of washing, drying, and folding a load before starting another load. Or you can use a pipelining type algorithm that when a resource is available and ready to be used you use it. All available resources are used when they can be and you get better throughput. As you can see in the second example if you use a resource when it is available in a correct manner you get more work done faster. Keeping this in mind.

Slide 26:

Richard:
This is a visualization of how pipelining is implemented in Camera2 api.

There are requests and a request queue.

When the resource, the camera is available it takes a new request from the que.

Camera2 api gets the settings and configurations of the camera, it then takes your request and creates a surface which is pretty much a template of everything that specific request wanted.

Finally the picture is taken.

So lets say i wanted my picture to be black and white, camera2 api would create that black and white surface and take the picture.

The picture is given back to us and a new request is taken.

However the lifecycle of the previous image is not finished and i can still edit the picture more such describing the width and height to save it as.

Slide 29:
Richard:
Now ill briefly explain how I was able to take a picture without user interaction. First thing first android requires that a preview must be created before any picture is taken. This was not helpful to my app because I don't want to give away that a picture is being taken. Steps I took to resolve this issue was to make the preview the smallest as possible so that you cant tell that the camera is open. So i made the preview be a 1 pixel by 1 pixel texture view and you would never know the camera is open and creating a preview. For demonstration purposes I made the camera preview take up the whole screen so you can see the camera is on.

Slide 30:
Richard:
Now that I got a preview that a camera can display on I get available cameras. In my code I have getCameraIdList()[0] because that is the back camera. If i wanted to access the front camera all i had to do was change the id to 1. Now that I have the camera and the preview.

Slide 31:
Richard:
I link what the camera is seeing to the preview.

Lines 286 to 290 is me getting the preview view configuration such as the width and height.

Line 301 is when I connect the camera and the preview.

And as soon as I connect the camera and the preview I call my take picture function.

The camera can be opened and working fine. But if you dont link the camera to a preview any take picture function that you created will fail. That is why as soon as I am legally allowed to take a picture after linking the camera and the preview I send the take picture request without the user needing to do anything. That is how I was able to take the picture without any user interaction.

Slide 32:
Richard: That was the end of our presentation!