# VERIFEYE

Modernize Security

Sunny Patel Samantha Husack Ethan Wallace Alexander Hurst

## The Problem

**In a nutshell: Traditional Security Systems suck for users**

# The Problem(s) with Traditional Systems

Single Point of Access

Not extensible without $$$
(neither hardware or software)

Often not accessible off-premises

# The Problem(s) with Traditional Systems

Single Point of Access

Not extensible without $$$
(neither hardware or software)

Often not accessible off-premises

## The Problem(s) with Traditional Systems

Single Point of Access

Not extensible without $$$
(neither hardware or software)

Often not accessible off-premises

## The Solution

**En un mot: VerifEye**

# VERIFEYE

A security system that is:

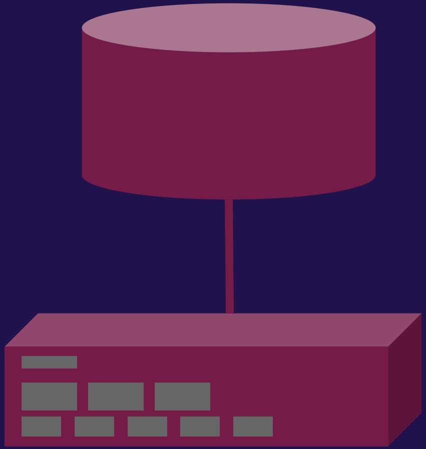
- Distributed
- Multi-platform
- Portable
- Extensible in hardware (supports addition of any number of cameras)
- Extensible in software
- O P E N  S O U R C E

# How VerifEye Works

## Client Devices
Display Footage

Available on IOS and Android

## Server and DB
Heavy Lifting



With non-proprietary, modifiable protocols

## Cameras
Put the Eye in VerifEye

Of any shape, size and number

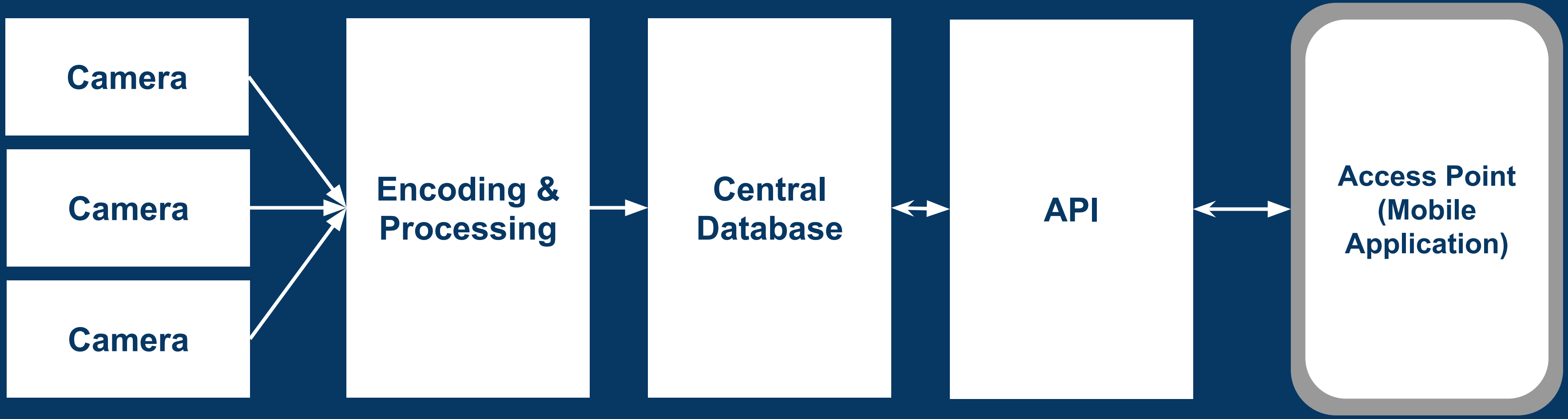
## The Design

A thorough overview

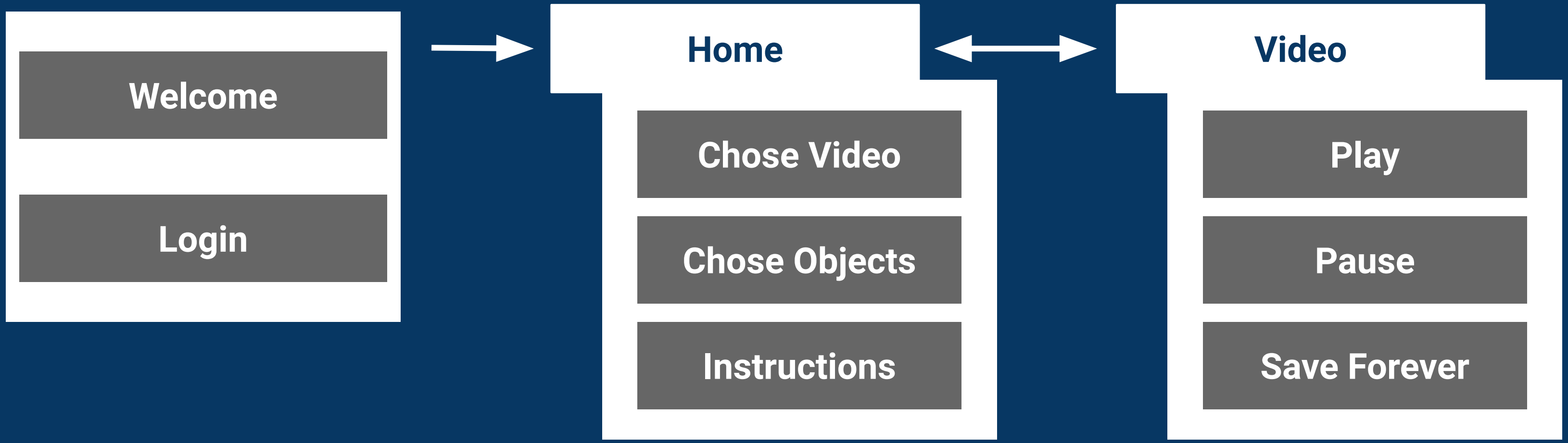
## Requirements

- ❑ Work on IOS and Android devices
    - ❑ Supporting Android 8.0 and above, targeting 10.0
- ❑ Support HEVC and H264 Encoding
- ❑ Security features to prevent unauthorized access
- ❑ Allow for multiple sites for a single account
- ❑ View footage from server

## Design: The System



## Design: User Interaction Flow



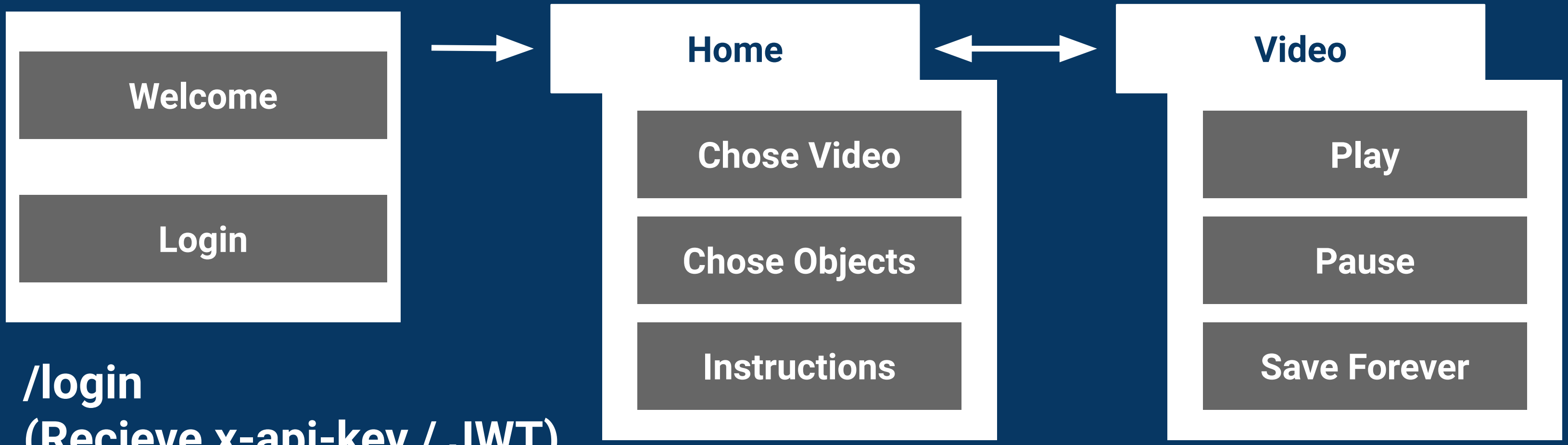
## Design: API Endpoints

- Java-based vert.x REST API
- Asynchronous, distributed event bus
- Very scalable architecture
- Many built-in structures

## Design: API Endpoints

- Easy to integrate services like JWT and OAuth
- Video sources are unique to different *organizations*
  - Users belonging to that organization have access to all assigned cameras
- Video data is chunked and sent asynchronously as a byte stream
  - Vert.x is able to stream video very efficiently by bypassing userland

# Design: API Endpoints



/login
(Recieve x-api-key / JWT)

/sources
x-api-header: key
token: JWT

/source
x-api-header : key
token: JWT

## The Implementation

**Behind the scenes**

## Flutter vs Traditional Native

**VerifEye was built with Flutter. Flutter is a dual-platform app development framework by Google**
**Here’s why we used Flutter instead of Android Studio**

### FLUTTER

- Minimize duplicated effort
- Uses Dart
- Built for production
- Intuitive UI management

### Traditional Native Android

- Minimize duplicated effort
- Uses Java
- Built for production
- Layout management somehow worse than HTML

## Implementation Details

Flutter is not like standard Android Development in Java

We used the BloC Pattern to separate Presentation and UI from core Business Logic

Events trigger State Changes

State Changes are tracked to activate Routes

A UI is rendered based on Routes

Similar to Basic Views, Flutter Widgets represent Layout Components

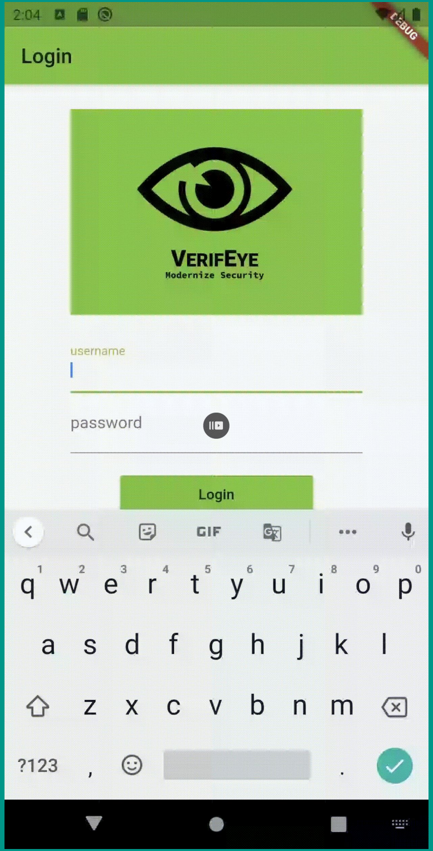
# Revisiting Project Requirements

| | |
|---|---|
| User Interface | Done by using Widgets and the Routing Service that Flutter provides |
| Multiple Activities | Implemented virtue of how Flutter renders UI |
| Intents | Changes between UI states are handled by the BloC routing service |
| State saving | State is withheld due to the BloC architecture and is seperate from the UI entirely |
| Internet Resources | Our API enables the user to download video files |
| Local Databases | Our user preferences are saved locally, but not explicitly |
| Centralized Database | Our API enables the app to pull from a centralized database |
| MultiThreading | Flutter handles different Routines and Services in separate threads from the UI Thread |

## Demonstration

Prepare thyselves...

# Demonstration



We shall now take questions